POLITECNICO
MILANO 1863

Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

EVENT-BASED VISION: BIO-INSPIRED APPROACHES FOR OBJECT RECOGNITION AND ROBOT CONTROL

BIO-INSPIRED APPROACHES BASED ON SPIKING NEURAL NETWORKS AND RL-INSPIRED MECHANISMS.

RELATORE:
Matteo Matteucci

TESI DI LAUREA MAGISTRALE DI:
Rafael Mosca
Matr. 919963

Anno Accademico 2019-2020

Rafael Mosca: *Event-based vision: bio-inspired approaches for object recognition and robot control* , bio-inspired approaches based on spiking neural networks and RL-inspired mechanisms. © Spring-Autumn 2020

SUPERVISORS:
Prof. Matteo Matteucci | Prof. Jean Martinet

UNIVERSITY:
Politecnico di Milano | Université Côte d'Azur

DEPARTMENT:
Scuola di Ingegneria Industriale e dell'Informazione

DEGREE COURSE:
Corso di Laurea Magistrale in Ingegneria Informatica

LOCATION:
Milan, Italy | Sophia Antipolis, France

TIME FRAME:
Spring-Autumn 2020

LICENSE:
CC BY-SA 3.0 Unported

KEYWORDS:
Event-based cameras, AER sensors, Bio-inspired, Spiking Neural Networks, SNN, R-STDP, Neuroscience, Reinforcement Learning

Part of this work has been supported by the French government, through the EUR DS4H Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-17-EURE-0004.

ABSTRACT

The human brain is able to perform amazing things, and even though researchers have tried to replicate its capabilities with Artificial Neural Networks, we are still very far away. One of the main differences between our artificial models and a real biological brain, is the fact that the brain is able to process vast amounts of information with amazing energy efficiency, this is very unlike our models which are very power hungry and sometimes require to be trained countless hours on high end machines. The problem with current models is that they require vast amounts of energy not only for training, but also for prediction. This is particularly critical in applications where we have limited power supplies such as drone navigation or IOT applications or where fast real-time computations are critical success factors.

In order to reduce the efficiency gap between artificial models and biological ones, researchers have tried to mimic the biophysical mechanisms of the brain in a better way than previous models, having as a goal a model which is more powerful and energy efficient than previous ones. This led to the third generation of ANNs: Spiking Neural Networks (SNNs). The search for efficiency in our artificial models did not limited itself just to the processing unit, it also extended to other areas such as image sensors, where new image sensors models like Event-Based camera sensors are inspired more on the biophysical mechanisms of the eye and the visual cortex, and are more efficient than ever before.

Despite being highly efficient, these new models work differently from traditional ones due to the addition of the temporal aspect and their asynchonicity, hence the algorithms that have been developed for previous models cannot be directly applied. We are therefore in need of new algorithms that are made *ad-hoc* for these models.

Although lot of progress has been made in several areas, there is still a lot to research to conduct. In particular, Reinforcement Learning and bio-inspired mechanisms have been largely unexplored and unexploited, hence this thesis aimed to understand whether these kind of algorithms can be applied to these new models and if so in what way. This thesis presents an analysis of how the different bio-inspired technologies (reinforcement learning, event based cameras and spiking neural networks) are related and can be used together. In particular we develop a new approach for object recognition and perform some novel experiments on robot control.

ACKNOWLEDGEMENTS

This thesis was carried out as part of a EIT Digital double degree master programme with Politecnico di Milano and Université Cote D'Azur. So first of all I would like to thank my two supervisors: Prof. Matteucci (POLIMI) and Prof. Jean Martinet (UCA) for the critical remarks, guiding and supporting me and for transmitting me the the passion for neuromorphic technologies and research in general.

I would also like to thank my family for all the emotional and economic support during my whole Master's Degree and my friends and peers Gabriele, Marina, Michele, Vittorio for all the moments we shared together during this journey which lasted 5 years.

I also thank, of course, Milad Mozafari and Zhenshan Bing whose papers were the a source of inspiration and starting point for this master thesis work, and Prof. Dr.-Ing. André Miede for providing for free the great *classicthesis* LaTeX-Template which I used for this thesis. And at last I would like to thank every professor and staff member at both universities which helped me be who I am today.

CONTENTS

List of Figures viii
List of Algorithms x
List of Code Snippets xi
List of Acronyms xii
1 INTRODUCTION 1
1.1 Thesis Outline 4
I BACKGROUND 6
2 SPIKING NEURAL NETWORKS 7
2.1 Artificial Neural Networks 7
2.2 Spiking neuron models 10
2.2.1 Hodgkin-Huxley Model 11
2.2.2 Leaky-Integrate and Fire Model 12
2.2.3 Izhikevich's Neuron Model 13
2.2.4 Spike Response Model 14
2.3 Information coding 15
2.3.1 Rate coding 15
2.3.2 Temporal coding 16
2.4 Learning Methods for SNN 17
2.4.1 Unsupervised learning 17
2.4.2 Supervised learning 19
2.5 Information decoding 20
2.5.1 Lateral Inhibition 20
2.5.2 Homeostasis 20
2.6 Neuromorphic Processors 21
3 EVENT-BASED VISION 22
3.1 Event-based Cameras 22
3.2 Address Event Representation 25
3.3 Event segmentation 26
3.4 Event representations 27
4 REINFORCEMENT LEARNING 28
4.1 Introduction 28
4.2 Bellman Equations 32
4.3 Generalized Policy Iteration 34
4.4 Reinforcement Learning Methods 36
4.5 Off-policy learning 42
4.5.1 Monte Carlo Importance Sampling 42
4.5.2 SARSA Importance Sampling 42
4.5.3 Q-learning 43
4.6 Value Function Approximation 43
4.6.1 Linear Value Function Approximation 44

4.6.2 Monte-Carlo Value Function Approximation 44
4.6.3 Temporal-Difference Value Function Approximation . . 45
4.6.4 Control using Value Function Approximation 45
4.6.5 Deep Q-Networks 45
4.6.6 Double DQN 46
4.6.7 Prioritized Replay 47
4.6.8 Dueling DQN 47
4.7 Policy Gradient 48
4.7.1 REINFORCE 50
4.7.2 Actor-Critic 50
4.7.3 Deep Deterministic Policy Gradient 51
4.7.4 Trust Region Proximal Optimization 52
4.7.5 Proximal Policy Optimization 53
4.7.6 Soft Actor Critic 53
4.8 Imitation Learning 54
4.8.1 Behavioural Cloning 55
4.8.2 Inverse RL 55
4.9 Recap 57

II CONTRIBUTIONS 58

5 BIO-INSPIRED TRINITY 59
5.1 Linking event-based cameras to SNNS 60
5.1.1 The object recognition link: HMAX 60
5.1.2 The motion estimation link: Reichardt detectors 62
5.2 Linking SNN to RL 64
5.2.1 Three-factor Rules 64
5.2.2 Reward-modulated STDP 65
5.2.3 Stochastic synapses 67
5.2.4 Basal Ganglia 69
5.3 Linking event-based cameras to RL 70
5.4 Linking the trinity 70

6 FEATURE EXTRACTION AND OBJECT RECOGNITION 74
6.1 CONTEXT AND STATE OF THE ART 74
6.1.1 EDHMAX 75
6.1.2 HFIRST 76
6.1.3 BOE 77
6.1.4 HOTS 78
6.1.5 MuST 80
6.1.6 Towards a new method 80
6.2 Our new approach: ReMuS 83
6.2.1 Pseudo-frame creation 84
6.2.2 Pseudo-frame processing 85
6.2.3 Spike encoding and multiscale fusion 86
6.2.4 SNN 87
6.2.5 An RL approach? 90
6.2.6 Experiments and results 90

6.2.7 Going beyond classification: object detection 96
6.2.8 Going faster: event-driven prediction 98
6.2.9 Discussion 98
7 ROBOT CONTROL 99
7.1 CONTEXT AND STATE OF THE ART 99
7.2 UAV robot control 101
7.2.1 The standard RL agents 103
7.2.2 The SNN agent 103
7.3 Experiments 104
7.3.1 The simulator: CoppeliaSim 105
7.3.2 The UAV 111
7.3.3 ROS 112
7.3.4 NEST network 113
7.3.5 The action model 115
7.3.6 The simplifications 116
7.3.7 The environment 117
7.3.8 Experiment results 121
8 CONCLUSIONS AND FUTURE WORK 127
8.1 Conclusions 127
8.2 Future work 128
8.2.1 RSTDDP 128
8.2.2 OBJECT RECOGNITION 128

III APPENDIX 130
A AERTB 131
B TAR 133
B.1 The algorithm 133
B.2 Results 134
B.3 Considerations 135
B.4 A new method 136
C NON-BIOINSPIRED PROCESSING APPROACHES 137
C.1 Processing events 137
C.2 Approaches for object detection 139
C.3 Approaches for robot control 139
C.4 What lies ahead? 140
D DATASETS 142
D.1 N-MNIST 142
D.2 POKER-DVS 143
D.3 DVS-BARREL 144

BIBLIOGRAPHY 145
E CVPR 155
F ICRA 164

LIST OF FIGURES

Figure 1 First generation ANN: Perceptron neuron. . . . 8
Figure 2 Biological model of a neuron. 8
Figure 3 Second generation ANN: Sigmoid neuron. . . . 9
Figure 4 Third generation ANN: Spiking neuron. 9
Figure 5 Synapse between neurons. 10
Figure 6 Representation of the neuron membrane. . . . 11
Figure 7 Electrical model of the Hodgkin-Huxley neuron model. 12
Figure 8 Electrical model of the LIF neuron model . . . 13
Figure 9 Dynamics of a LIF Neuron 14
Figure 10 A common STDP window 18
Figure 11 Some other STDP windows. 19
Figure 12 Working principle of an event-based camera . 23
Figure 13 Inner structure of the human retina 23
Figure 14 Standard vs. Event Camera Image 24
Figure 15 DAVIS sensor 25
Figure 16 The AER communication protocol. 25
Figure 17 Hard Event Segmentation. 26
Figure 18 Soft Event Segmentation 26
Figure 19 General event-based representation framework 27
Figure 20 The standard RL paradigm. 29
Figure 21 Example of an MDP 33
Figure 22 Generalized Policy Iteration 35
Figure 23 A graph of the G_t^λ weighting function 40
Figure 24 Value function approximation 43
Figure 25 Examples of VFAs 43
Figure 26 Dueling DQN Architecture. 48
Figure 27 General Actor-Critic vs DDPG architecture. . 51
Figure 28 Triad of approaches and their intersections. . . 59
Figure 29 Neuromorphic hardware and their biological counterparts 61
Figure 30 HMAX model 62
Figure 31 Coincidence detector with synaptic delays . . 63
Figure 32 Spike-Timing Dependent Delay Plasticity . . . 63
Figure 33 Release of neurotransmitters. 65
Figure 34 Dopamine Modulated STDP 67
Figure 35 Basal Ganglia. 69
Figure 36 Biological vs neuromorphic control loop 71
Figure 37 Human action mechanism. 72
Figure 38 EDHMAX Architecture. 75
Figure 39 HFirst Architecture. 76
Figure 40 Time-surfaces 78

Figure 41 Overview of the ReMuS approach. 83
Figure 42 A threshold-based Motion Symbol Detector (MSD). 84
Figure 43 A reconstructed pseudo-frame. 85
Figure 44 Gabor filters. 86
Figure 45 C1 Layer. 86
Figure 46 Feature encoding 87
Figure 47 Feature encoding functions 87
Figure 48 S2 Grid 88
Figure 49 Pattern comparison with different receptive field sizes 92
Figure 50 Learnt grid prototypes on the POKER-DVS dataset 92
Figure 51 Different prototypes for a same class 93
Figure 52 Learned Grid Prototypes on the N-MNIST dataset 94
Figure 53 Object detection with our proposed ReMuS approach 96
Figure 54 Multi-object detection 97
Figure 55 The general simulation framework for robot control. 100
Figure 56 RL Frameworks for standard and SNN agents. . 102
Figure 57 SNN agent network overview. 103
Figure 58 The general design framework for robot control 104
Figure 59 Software stack overview. 105
Figure 60 CoppeliaSim 106
Figure 61 Simulation Loop 107
Figure 62 Overview of the ball-following task. 108
Figure 63 Ball-target scenario 108
Figure 64 Overview of the line-following task. 109
Figure 65 Line-following scenario 109
Figure 66 Simulated DVS camera model 110
Figure 67 Model of the drone 110
Figure 68 The UAV model 111
Figure 69 Movements of a quadricopter 111
Figure 70 ROS Topics 112
Figure 71 Nest simulator logo 113
Figure 72 SNN Network 114
Figure 73 The action model for the SNN agent. 115
Figure 74 Remote API files 117
Figure 75 Bounding Box 119
Figure 76 State Representation 120
Figure 77 Simulation results for a ball-target scenario. . . 123
Figure 78 Results for the SNN-R-STDP agent. 124
Figure 79 Results for the DQN agent. 124
Figure 80 Results for the line-following scenario. 125
Figure 81 Sobel Filters 129
Figure 82 `aertb`: custom shell 132
Figure 83 TAR algorithm example 134

Figure 84 Differences between filtered images and not . 134
Figure 85 Impact of Noise filtering on N-MNIST 135
Figure 86 Result of an object classification TAR method . 136
Figure 87 Common frame-integration methods of events used with deep learning approaches. 138
Figure 88 The fully-convolutional YOLE architecture . . . 139
Figure 89 Steering-angle prediction architecture 140
Figure 90 Data capture method for the N-MNIST dataset . 142
Figure 91 A digit in the N-MNIST dataset 142
Figure 92 POKER-DVS dataset 143
Figure 93 Data capture method for the DVS-BARREL dataset 144
Figure 94 Samples of the DVS-BARREL dataset. 144

LIST OF ALGORITHMS

1 Iterative policy evaluation 35
2 Value iteration 35
3 Every–Visit Monte–Carlo Policy Evaluation 36
4 First–Visit Monte–Carlo Policy Evaluation 37
5 SARSA(λ) 41
6 Deep Q-learning with Experience Replay 46
7 Double Q-learning 47
8 REINFORCE, Monte-Carlo Policy-Gradient Method 50
9 Actor-Critic Algorithm 50
10 DDPG 52
11 DAGGER: Dataset Aggregation 55
12 Apprenticeship Learning 56
13 Online Clustering of Time-Surfaces 79
14 Temporal Activity Region (TAR) 133
15 Leaky surface frame integration 138

LIST OF CODE SNIPPETS

1 Simple NEST Simulation 113
2 `env` loop 117
3 Publishers and Subscribers 118

4 `aertb`: loading events 131
5 `aertb`: using HDF5 132

ACRONYMS

AER	Address Event Representation
ANN	Artificial Neural Network
ATIS	Asynchronous Time-based Image Sensor
BG	Basal Ganglia
BOE	Bag Of Events
CD	Contrastive Divergence
CLI	Command Line Interface
CNN	Convolutional Neural Network
DA	Dopamine
DAVIS	Dynamic and Active-pixel Vision Sensor
DBN	Deep Belief Network
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
DVS	Dynamic Vision Sensor
DNN	Deep Neural Networks
EPSP	Excitatory Postsynaptic Potential
F2F	First to Fire
GLIE	Greedy in the Limit with Infinite Exploration
HDF5	Hierarchical Data Format version 5
HES	Hard Event Segmentation
HFR	Highest Firing Rate
I2L	Intensity to Latency
IF	Integrate and Fire
IMU	Inertial Movement Unit
IT	Inferior Temporal Cortex
IPSP	Inhibitory Postsynaptic Potential

LGN	Lateral Geniculate Nucleus
LIF	Leaky Integrate and Fire
LTP	Long Term Potentiation
LTD	Long Term Depression
M1	Primary Motion Cortex
MC	Monte-Carlo
MDP	Markov Decision Process
MLP	Multi Layer Perceptron
MSD	Motion Symbol Detector
NEST	NEural Simulation Tool
ODE	Ordinary Differential Equation
PFC	Prefrontal Cortex
PI	Policy Iteration
PPO	Proximal Policy Optimization
PSP	Post Synaptic Potential
ReMuS	Reward-modulated MultiScale
RL	Reinforcement Learning
ROS	Robot Operating System
R-STDP	Reward-modulated Spike-Timing Dependent Plasticity
R-STDDP	Reward-modulated Spike-Timing Dependent Delay Plasticity
SAC	Soft Actor Critic
SES	Soft Event Segmentation
SNN	Spiking Neural Network
SNc	Substantia Nigra pars compacta
SR	Segment Recorder
SRM	Spike Response Model
STDDP	Spike-Timing Dependent Delay Plasticity
STDP	Spike-Timing Dependent Plasticity

TAR	Temporal Active Regions
TD	Temporal Difference
TRPO	Trust Region Policy Optimization
UAV	Unmanned Air Vehicle
VFA	Value Function Approximation
V1	Primary Visual Cortex
V2	Secondary Visual Cortex
V4	Visual Area
VI	Value Iteration
V-REP	Virtual Robot Experimentation Platform
WTA	Winner-Take-All

1 INTRODUCTION

Researchers and engineers have found inspiration in nature countless times throughout history. This recurrent practice, which belongs to a field which is known as *biomimetics*, has provided the world with many innovations: from better aircrafts and trains, to robots and theoretical models which closely imitate nature to accomplish a certain task. What makes this field so attractive is the fact that all the characteristics of an animal or plant that we observe in nature, are not completely random, but are the result of centuries and centuries of optimisation through natural selection and evolution, thus, by imitating or getting inspired by what we see in nature, we can potentially increase efficiency or efficacy at performing a certain task.

However, not all bio-inspired inventions perfectly reflect their source of inspiration, such an example are Artificial Neural Networks (ANNs). The first generations of ANNs, are vaguely inspired in the brain, and despite being relatively powerful, they are extremely resource intensive. This significantly differs from the human brain. ANNs require a lot of computational power and considerable amounts of annotated data to be trained to perform a task. For instance, the well known ResNet architecture was trained for 3 weeks on a 8-GPU server [1], which is equivalent to a power consumption of about 1 GWh. The human brain on the other hand, is extremely efficient as it requires roughly 20W to function [2] and is capable of performing not one, but myriads of tasks, even more than one at the same time.

It is clear that we are still far from achieving the computational power and efficiency of the human brain, for this reason, researchers in the Computational Neuroscience field have been studying how the brain carries out cognitive functions and have tried to describe the principles that govern the development, structure, physiology and cognitive abilities of the brain with both simple and intricate mathematical models. The goal of computational neuroscientists is not merely descriptive, a model also needs to be reproducible so we can possibly take advantage of it. This field of study has given a lot of contributions to the world, but most importantly it has inspired what we call the "third generation of ANNs": Spiking Neural Networks (SNNs) [3].

Spiking Neural Networks (SNNs) are a type of ANN which has more fidelity to the biological model of a neuron as information is conveyed to other neurons by means of electrical pulses called *spikes* [4]. In this kind of neural network, the information can be processed asynchronously, it is more robust to noise, and it has even shown to be more computationally powerful than previous generations of ANNs [5].

Despite being far away from being able to model the entire human brain, today there are multiple research-purposed neuromorphic chips, i. e. , hardware that implements bio-inspired neuronal models like Spiking Neural Networks. Examples of these chips are: Intel's Loihi, IBM's TrueNorth, or SpiNN5 from the University of Manchester. These kind of chips are relatively small and are able to simulate roughly a hundred thousand neurons.

Neuromorphic processors have made SNNs very appealing as the processing of a single spike on these chips may only consume a few pJ of energy [6]. This fact potentially presents the possibility of building neural networks which are energy efficient and as it has been shown, potentially more computationally powerful.

Although SNNs present a lot of appealing characteristics, SNNs are not very popular yet and their use is confined to research labs, making their potential largely underexploited. This is mainly due to the fact that traditional algorithms for neural networks cannot be used as the underlying working principle of the neurons is substantially different and standard algorithms such as back-propagation [7] cannot be applied.

In the last few years there has been a surge in SNN algorithms, both supervised (example-based), and unsupervised (data-driven), which has increased their popularity, while approaches based on the reinforcement learning paradigm have not been explored thoroughly on these new kind of neural networks which makes it quite unusual as it is the area of machine learning that most closely represents how humans learn.

Differently form other paradigms, Reinforcement Learning (RL) is focused on how agents should take actions in an environment in order to maximise a reward. This paradigm results particularly useful when the dynamics of the environment are unknown or are difficult to be modelled, and it has been successfully used to tackle complex problems. The scope of this thesis is to explore whether if is possible to use RL-inspired approaches in SNN, in particular when event-based vision sensors are associated.

Event-based cameras, are biologically inspired vision sensors that do not collect frames at a fixed frame rate, instead, these kind of sensors output changes on pixel-level brightness. By doing so, these kind of cameras offer some considerable advantages over standard cameras, namely a very high dynamic range, avoidance of motion blur, and a latency in the microsecond range. These sensors emit events asynchronously which makes SNNs perfectly suited for their processing [8], [9]. Event-based cameras coupled with SNN are a perfect fit for autonomous robots or vehicles that need energy efficient and fast real-time calculations.

At the time this document is being written, it makes no sense to compare traditional deep learning approaches to bio-inspired approaches. Traditional Deep Neural Networks (DNN) outperform bio-inspired ones in terms of accuracy and other performance-related scores. This is due to many things: lack of large datasets collected with bio-inspired sensors, unavailability of neuromorphic hardware at mass scale, novelty of the neuromorphic field, and many other reasons. It is worth mentioning that spiking neural networks are relatively new compared to previous generations, traditional technologies have an advantage of 20-30 years of research.

These approaches and experiments will not be implemented on neuromorphic hardware since it is still unavailable at mass scale, instead, as many of current state of the art approaches, neurons will be simulated at software level. The objective of the research is thus to conceptually create new models and approaches which even if they do not possess all the qualities and improvements over standard approaches due to the nature of the software simulations, these approaches lay out the foundations of future approaches and implementations. Only when neuromorphic hardware becomes more widely available and less expensive is when we will be able to fully take advantage of the power efficiency and asynchronicity which they offer.

Since spiking neural networks, event-based cameras, and reinforcement learning are all inspired by biology itself, they may have an enormous untapped potential, especially if they can be used together.

To address this matter, the following questions were raised and paved the way of this work:

- Is there a relationship between Spiking Neural Networks (SNNs)-Reinforcement Learning (RL)-event-based sensors?
- Can we use RL on SNNs? What are the current approaches?
- Can we use the previous approaches on data captured by Event Based Cameras or do we need a new approach?
- Where can we use such a triad of approaches?
- Can we develop new methods using the triad of bio-inspired approaches?

1.1 THESIS OUTLINE

The thesis is organised in two main parts: background and contributions. The thesis was structured this way so that readers familiar with such technologies can skip Part I and head to Part II where the main contributions of this work are explained.

(i) Part I contains an introduction of each of the bio-inspired technologies.

  (1) Chapter 2 introduces Spiking Neural Networks, highlighting the difference with respect previous generations of neural networks, introducing common models, learning rules, as well as problems that have risen due to their novelty.

  (2) Chapter 3 introduces event-based vision sensors, their working principle, advantages, as well as common processing techniques.

  (3) Chapter 4 introduces reinforcement learning and goes into details of some approaches because many of the concepts will be useful and referenced in later chapters.

(ii) Part II contains the contributions of this master thesis

  (1) Chapter 5 links RL, SNN and event cameras, describe the links between each couple of technologies and how they all intersect, as well as the areas of interest where they can be applied together.

  (2) Chapter 6 introduces feature extraction and object recognition, an area of application where the three approaches can be used together, and propose, to the extent of our knowledge, what is the very first approach using all three technologies.

  (3) Chapter 7 introduces robot control, another area of application which can benefit from the three bio-inspired approaches, explains how such an approach is structured, and shows the results of some novel computer simulations aiming to determine how the performance of traditional RL approaches compares to approaches using SNNs.

(v) The appendices contain extra approaches, content and information relevant to the work.

  (1) Appendix A describes `aertb`, a PyPI Python library with a CLI interface created during the thesis work which aims to facilitate development with event-based vision data.

  (2) Appendix B introduces TAR, a simple noise filter algorithm, proposed for filtering noise on an actual robot implementation of the approach in Chapter 7, which lacks noise due to the nature of the computer simulations.

3. Appendix C contains additional information on non-bio-inspired approaches for event-based vision to better compare them with bio-inspired approaches.
4. Appendix D describes the event-based datasets used to test the object recognition approach described in Chapter 6.

Finally, at the end of this thesis there are two papers attached that were submitted to international conferences for their review. In particular the paper summarising the work in Chapter 6 has been submitted at the Conference on Computer Vision and Pattern Recognition (CVPR) 2021 under the category Neuromorphic sensors, while the paper summarising the work in Chapter 7 has been submitted to International Conference on Robotics and Automation (ICRA) 2021.

Part I

BACKGROUND

In this first part we describe the motivation of the work and lay the foundations so that the contributions in Part II can be fully understood. In particular in Chapter 2 we introduce Spiking Neural Networks and some of the existing learning methods, in Chapter 3 we introduce event-based camera sensors as well as some existing algorithms, while in Chapter 4 we introduce Reinforcement Learning very in depth as some concepts will be used in later chapters.

2 SPIKING NEURAL NETWORKS

2.1 ARTIFICIAL NEURAL NETWORKS

An Artificial Neural Network (ANN) is a computing system inspired by the structure and function of the human brain that over the last century has been used for multiple purposes, ranging from regression analysis and classification, to system identification and control.

ANNs are composed of artificial *neurons* which receive an input, translate the information received to a modification of its internal state with the help of an *activation function*, and produce an output.

Nowadays there are many typologies and architectures of neural networks, however, according to Maass [5] and Ghosh-Dastidar and Adeli [3], we can categorise ANNs in generations according to the information processing mechanism of the artificial neurons: the activation function. If we use this criterion, we can distinguish three generations of neural networks.

The first generation of neurons was proposed by McCulloch and Pitts [10] in 1943. These neurons, also referred to as perceptrons, are able to operate in a multi-layered network only with digital (binary) inputs and outputs. As weighted synapses convey signals to the neuron, the neuron changes its internal state u by performing a weighted summation of each input value x_i:

$$u = \sum_{i=0}^{n} w_i x_i \ . \tag{1}$$

The output is then processed with a Heaviside activation function f which outputs whether this internal value crosses a given threshold ϑ, emitting `1` if the threshold is crossed or `0` in case it is not crossed:

$$f(u) = \begin{cases} 1, & u \geqslant \vartheta \\ 0, & u < \vartheta \end{cases} \ . \tag{2}$$

This model of neuron gave rise to well-known models such as Boltzmann machines [11] and Hopfield nets [12]. A schematic of the information flow in this generation of neuron can be found in Figure 1.

It is easy to draw parallels between this model of neuron and its biological counterpart represented in Figure 2.

- the DENDRITES act as the input vector and allow the cell to receive signals from a large number of neurons.

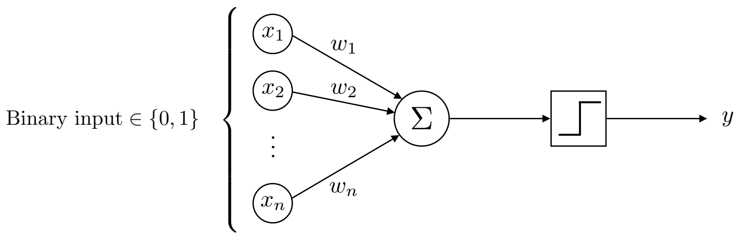


Figure 1: First generation ANN: Perceptron neuron.

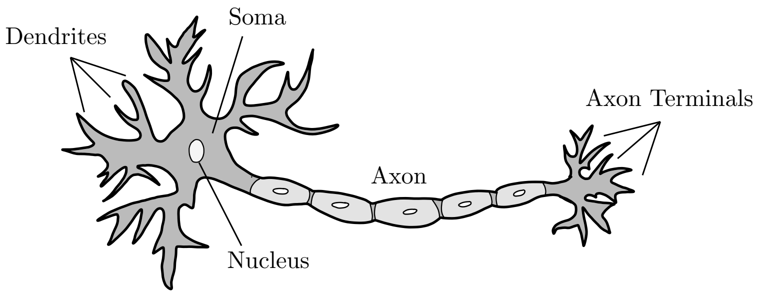


Figure 2: Biological model of a neuron.

- the SOMA, i. e. , the cell body of a neuron, acts as the summation function represented in Figure 1. As positive (exciting) and negative (inhibiting) signals arrive to the soma from the dendrites, the charged ions (+/−) are effectively added as a result of a mixture to the solution inside the soma, just like the summation does with the inputs.
- the AXON, i. e. , the nerve fiber that conducts action potentials away from the cell body, is what allows us to connect multiple neurons together.

The second generation of neurons developed from the 1950s to 1990s. As shown in Figure 3, this model of neuron replaces the Heaviside activation function for a non-linear function capable of handling real-valued inputs and outputs such as the *sigmoid function* $\sigma(u)$ or the *hyperbolic tangent function* $\tanh(u)$.

$$\sigma(u) = \frac{1}{1 + e^{-u}} \,, \qquad (3) \qquad \tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}} \,. \qquad (4)$$

This kind of neurons is typically used in feedforward and recurrent neural networks. This model has also gained a lot of popularity over time because it supports gradient descent learning algorithms such as back-propragation [7].

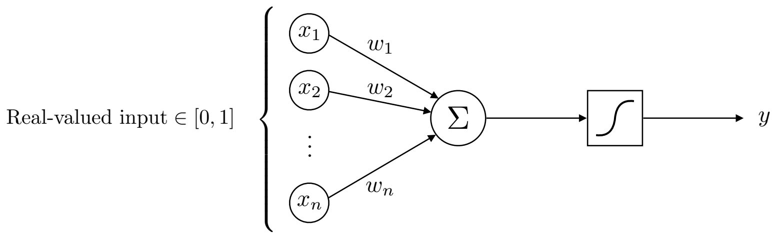


Figure 3: Second generation ANN: Sigmoid neuron.

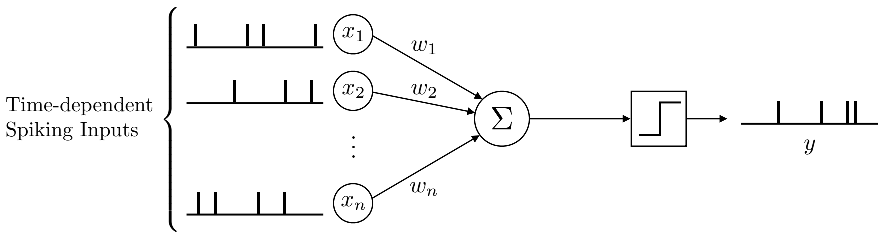


Figure 4: Third generation ANN: Spiking neuron.

Researchers tried to come up with more biologically precise models of a neuron that considered the temporal aspect of firing within the synapses. This aspect, which was not considered in previous models, led to the third generation of Artificial Neural Networks also referred to as Spiking Neural Networks (SNNs).

Spiking Neural Networks are a special type of artificial neural networks where neurons communicate by generating and propagating electrical pulses called *action potentials* or *spikes*. These spikes usually have a duration of a few milliseconds and an amplitude of around 100 millivolts [4]. A sequence of spikes in time is usually called *spike train* and is mathematically represented as the sum of the Dirac impulses at firing times $t^{(f)}$:

$$S(t) = \sum_{f} \delta(t - t^{(f)}) \ . \tag{5}$$

In this model, shown in Figure 4, when a neuron fires, the neurons at the other end (post synaptic neurons) are affected and their internal variable state (*membrane potential*) changes, making the neuron who received the pulse more or less likely to fire for some duration of time. This transient impact on the other neurons which is also called *postsynaptic potential*, can thus have two effects: inhibit the firing – Inhibitory Postsynaptic Potential (IPSP) – or excite the neuron – Excitatory Postsynaptic Potential (EPSP).

Neurons connect and communicate with one another through specialised junctions called *synapses*. Normally these synapses are weighted by *synaptic weights* and may have a *synaptic delay*. A schematic with

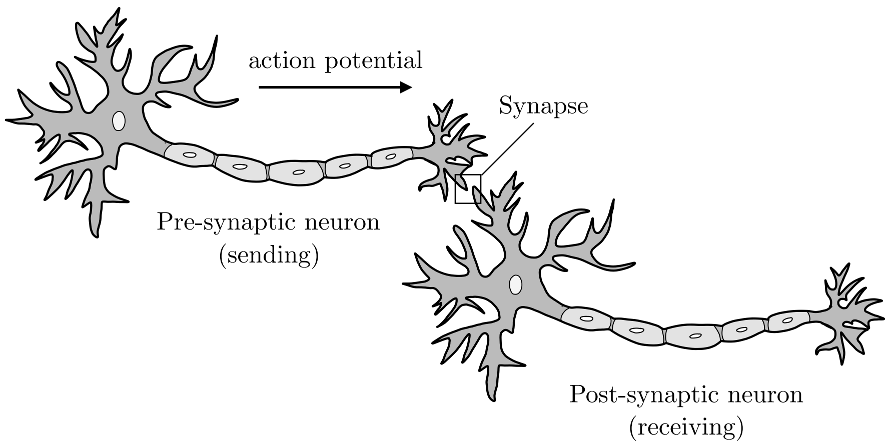


Figure 5: Synapse between neurons.

common terminology in this kind of networks can be found in Figure 5.

The human brain is composed of a 100 billion neurons and more than 150 trillion synapses [13]. Each of these synapses are strengthened or weakened, and the modifications to these synapses lay the foundations of learning and memory. As we will see in Section 2.4.1.1, this mechanism, also known as synaptic plasticity, can be simulated on Spiking Neural Network (SNN) and can be used to learn to perform certain tasks.

What makes SNNs so interesting, is the fact that it is not only a conceptual/theorical model, in fact, it can be implemented on what we call neuromorphic hardware. This particular kind of hardware, which can be built with CMOS technology, typically uses low power (under the threshold voltage), and is capable of reducing energy dissipation by several orders of magnitude compared to standard digital architectures [14], therefore opening the doors to potentially powerful and low-powered applications.

It has been demonstrated that these models are computationally more powerful than first and second generation counterparts [5], hence this model can be in theory applied to all problems solved by previous models, but also opening the doors to fast and efficient solutions to problems such as signal-processing and event detection.

2.2 SPIKING NEURON MODELS

Spiking Neural Network do not describe a single model but rather an ensemble of models which operate with spikes and have the characteristics mentioned in the previous section. Inside this ensemble we have very simple models such as the Integrate and Fire (IF) and the Leaky Integrate and Fire (LIF), and very complicated models like the

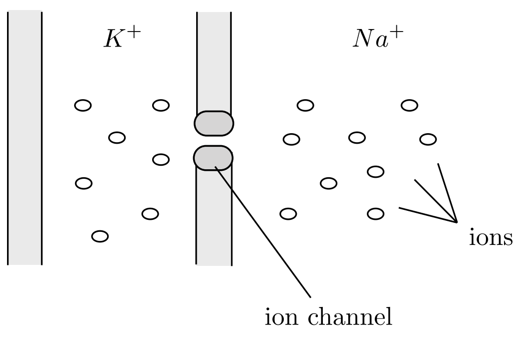


Figure 6: Representation of the neuron membrane.

Hodgkin-Huxley model or the Izhikhevic model. Each one of these models has some characteristics which make it better suited for some tasks rather than others. If we had to summarise the differences between the models we could say that the main differences lie on the biological accurateness and the simplicity of simulation.

In this section, we will briefly introduce the most common SNN models so algorithms that can be found in future sections and rely on such models, can be properly understood.

2.2.1 *Hodgkin-Huxley Model*

As it was mentioned before, spiking neural networks are based on a model of neuron that resembles more to the biological neuron and considers their spiking nature. Work on this matter dates back to the 1950s when Hodgkin and Huxley [15] presented a model which describes, through a set of nonlinear differential equations, how action potentials (spikes) in neurons are initiated and propagated. This model was so detailed and revolutionary that the authors were awarded a Nobel prize in 1963.

This model considers that the neuron, as any other cell, is surrounded by a cell membrane that separates the inside of the cell (*intracellular*) and the outside of the cell (*extracellular*). Embedded in the cell membrane are *ion channels*, that can open or close a pore through which ions can flow inside and outside the cell, and *ion pumps*, that use energy to push sodium ions (Na^+) outside the cell and pull potassium ions (K^+) inside the cell (Figure 6). The concentration difference of ions gives a voltage difference (also called *reversal potential*) which is described by the Nernst equation:

$$\Delta u = -\frac{kT}{q} \ln \frac{n(u_{in})}{n(u_{out})} , \tag{6}$$

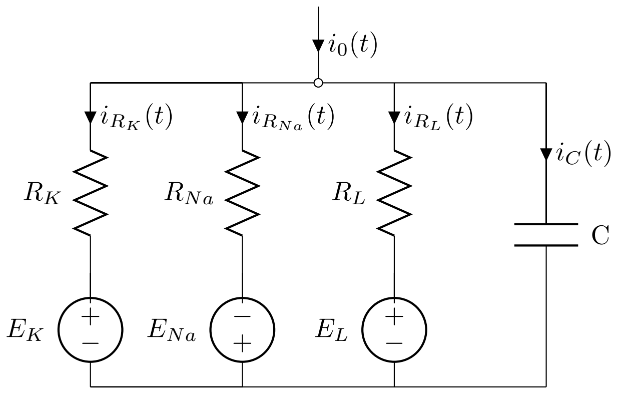


Figure 7: Electrical model of the Hodgkin-Huxley neuron model.

where k is the Boltzmann constant, T is the temperature in kelvins, q is the electrical charge, and $n(u_{in})$ and $n(u_{out})$ are respectively the relative concentration of a certain ion inside and outside the neuron.

This relation holds for every ion type, hence the total equilibrium potential is an equilibrium between the reversal potential of sodium, potassium and other ion types.

Even though this model is very realistic can be easily modelled as an electrical circuit consisting of capacitors and resistors that model different ion channels: sodium, potassium, and other ion types (Figure 7), it is too complex to simulate and it would be intractable to compute temporal interactions in large networks of neurons. Today it is used more for an accurate description of the processing performed by neurons rather than a useful model used for practical applications.

2.2.2 *Leaky-Integrate and Fire Model*

Today most models of spiking neurons tradeoff the accurateness of biophysical mechanisms that contribute to the formation of a spike for a more computationally tractable model. One of the main exponents of such models is the Leaky Integrate and Fire (LIF) model where the dynamics of the membrane potential in a neuron are described by a simpler, single first order Ordinary Differential Equation (ODE):

$$C\frac{du}{dt} = -\frac{1}{R}u(t) + \left(i_o(t) + \sum w_j i_j(t)\right) , \tag{7}$$

where C is the membrane capacitance, R is the input resistance, w_j represents the weight of the j-th synapse, $i_j(t)$ the input current from the j-th synaptic input and $i_o(t)$ the external current. Roughly speaking, in this model of neuron, the neuron suffers an increase in its internal state by a weighted product of the input currents. The neuron suffers however of *leakage*, hence the name, meaning that the internal

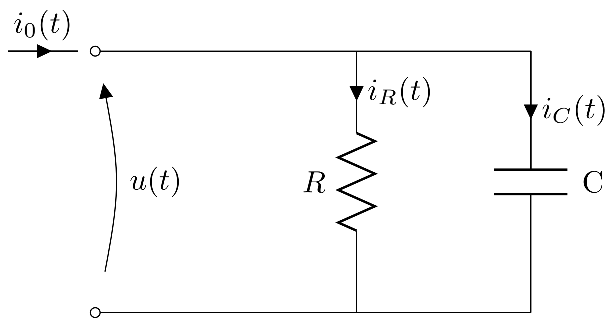


Figure 8: Electrical model of the leaky integrate and fire neuron model.

state value decays exponentially with a time constant given by model parameters.

As shown in Figure 8, this model can be seen as a parallel RC circuit that models the voltage leakage on the neuron with a time constant of $\tau_{leak} = RC$. By applying Kirchhoff's current law, we can write $i_o(t) = i_R(t) + i_C(t)$, applying Ohm's law to $i_R(t)$ we obtain $i_R(t) = \frac{u(t)}{R}$ and expressing $i_C(t)$ with the dynamic expression $i_C(t) = C\frac{du(t)}{dt}$ we can easily obtain the ODE in Equation 7.

In this model, a neuron fires a spike at time $t^{(f)}$ whenever the state variable $u(t)$ — that corresponds to the membrane potential — crosses a certain threshold ϑ as expressed by Equation 8. It is important to notice that we also need the condition $u'(t^{(f)}) > 0$ as we want to send a spike only when the threshold is reached from below and not when the membrane potential crosses the threshold from above as a result of leakage.

$$o(t^{(f)}) = \begin{cases} \text{fire} & \text{iff} \quad u(t^{(f)}) = \vartheta \text{ and } u'(t^{(f)}) > 0 \\ \text{do not fire} & \text{otherwise} \end{cases} \tag{8}$$

Immediately after sending an output spike, the neuron enters a period of reset or "refractoriness" which sets the membrane potential to a predefined reset value or 0. It is important to note that this model totally ignores the shape of the action potential as the firing of a spike is defined only by the time the threshold is reached as seen in Figure 9.

The Integrate and Fire (IF) model is a further simplification of the LIF model which ignores the leakage of the membrane potential. This can be modelled as the RC circuit in Figure 8 in which the resistance $R \to \infty$.

2.2.3 *Izhikevich's Neuron Model*

As we saw in the previous subsections, the Hodgkin–Huxley neuron model is very accurate but computationally prohibitive, whilst LIF is

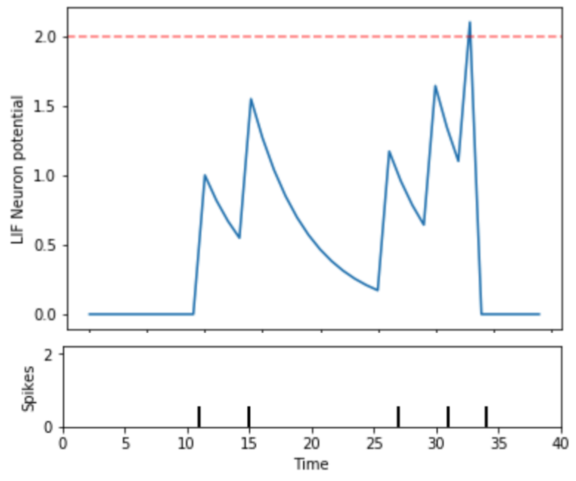


Figure 9: Dynamics of a LIF Neuron with a given spike train.

simple and computationally tractable. However the LIF model is too simple and incapable of reproducing many behaviours that can occur in biological neurons.

To solve this problem we can use a model developed by Izhikevich [16] which is characterised by two differential equations that represents a good trade-off between computational tractability and biological accurateness.

The first differential equation (Equation 9) focuses on the membrane potential while the second one (Equation 10) represents a membrane recovery variable which accounts for the activation of currents of K^+ ions and inactivation of currents of Na^+ ions, and thus provides a negative feedback to the membrane potential.

$$du/dt = 0.04u(t)^2 + 5u(t) + 140 - w(t) + i_o(t) \ , \tag{9}$$

$$dw/dt = a(bu(t) - w(t)) \ , \tag{10}$$

with after spike resetting: if $u \geqslant \theta$ then $u \leftarrow c$ and $w \leftarrow w + d$.

This model can reproduce the 20 main naturally occurring firing schemes while the LIF model can reproduce only 3 of them. Among the covered patterns include sub-threshold oscillation, tonic spiking, and phasing spiking, just to name a few.

2.2.4 *Spike Response Model*

The Spike Response Model (SRM) is a neuron model developed by Gerstner and Kistler [17] that generalises the Leaky Integrate and Fire model. It still generates a spike when the action potential passes from below, but unlike the LIF model, the threshold θ is not constant, instead, it is based on the instant the last spike was fired. Since typically the threshold in this model increases after receiving a spike and then decays to a resting value after some time, this can be used to model *refactoriness*. The action potential is given by the equation

$$u_j(t) = \eta(t - \hat{t}_j) + \sum_i \sum_f w_{ij} \epsilon_{ij}(t - \hat{t}_j, t - t_i^f) \ , \tag{11}$$

where w_{ij} is the weight of the synapse connecting the presynaptic neuron i to the post-synaptic neuron j, $\hat{t}_j$ is the firing time of the last spike of the post-synaptic neuron j, t_i^f the arrival time of the f-th spike at presynaptic neuron i , η is a function modelling the potential reset after firing a spike, and ϵ is the membrane potential's response to presynaptic spikes.

2.3 INFORMATION CODING

In the last section we covered several spiking neuron models which process an incoming spike train. However we have not talked about how the information is conveyed through spikes. This depends on the selected *coding scheme*, that is, the way information is encoded in a spike train. There are mainly two coding schemes: *rate coding*, which depends only on the average number of spikes per time unit, and *temporal coding* which depends on the precise timing of single spikes. There are also some other coding schemes such as *population coding* and *sparse coding*, however they are not commonly used. There has been a strong debate between researchers regarding which information coding must be used.

2.3.1 *Rate coding*

Rate coding, sometimes called *frequency coding* is a coding scheme that assumes that most, if not all information about the stimulus, is contained in the firing rate of the neuron. A popular coding mechanism is the spike-count rate. This approach, also referred to as temporal average, is obtained by counting the number of spikes during a certain time interval and by dividing this number with the duration T of the time interval to obtain the firing intensity λ of the neuron:

$$\lambda = \frac{n_{spikes}}{\Delta T} \ . \tag{12}$$

This kind of rate coding accepts a Poisson model as an appropriate description:

$$\Pr\{ \text{ k spikes in interval T}\} = \frac{\lambda^k}{k!} e^{-\lambda} \ . \tag{13}$$

2.3.2 *Temporal coding*

In *temporal coding* it is assumed that the information about the stimulus is contained in the specific precise spike timing of the neuron.

The ongoing debate between the coding schemes has had arguments supporting both schemes, however, it has been observed that neurons in the visual cortex precisely respond to a stimulus on a millisecond timescale, hence supporting the temporal coding scheme [18], in additional strong arguments against rate coding have been given by Thorpe et al. [19] in the context of visual information processing.

Sometimes, the information that needs to be processed by a SNN is not always in a spike format, e. g. , images, so in order to achieve such representation we have to use one of the several methods that allow the conversion of a of real number to a spike with a precise firing time (also called timestamp).

Given a real-valued input r with maximum possible value r_{max} and minimum possible value r_{min} , it is possible to associate the input to a specific firing time ts with the following approaches:

- Intensity to Latency (I2L): this approach uses the magnitude of the input value (also called *intensity*) to generate a spike at a particular timestamp. In such method the spike time is inversely proportional in a linear way to the intensity of the input value, i. e. , giving earlier timestamps to higher values and later timestamps to lower values. Such timestamps are usually distributed within a fixed time window t_w with the following formula:

$$ts(r) = t_w * \frac{r - r_{min}}{r_{max} - r_{min}} \ . \tag{14}$$

- Logarithmic I2L: this approach works in a very similar way to I2L, but instead of proportionally distributing the timestamps in a certain time window, this approach uses a mapping function that is logarithmic instead of linear:

$$ts(r) = t_w * \frac{\ln(r_{max}) - \ln(r)}{\ln(r_{max}) - \ln(r_{min})} \ . \tag{15}$$

- Rank: this approach does not consider the actual magnitude of the value, but rather considers its position with respect to the other values which are being processed together. The highest value is given a timestamp of 1 representing the first spike time, the second highest value is given a timestamp of 2, and so on. There is no need to fix a spiking time window, the order is given by the sorting of the responses and the last timestamp equals the number of responses. If a time window is fixed, one can introduce a delta between spikes which is equal to $1/n_{spikes}$. Note that it is quite different from I2L since the delta between values is not necessarily represented with a proportional difference in spike times.

There has also been some discussion regarding which conversion is better than the others with recent work presenting strong arguments supporting log I2L [20]. However, it is not uncommon to see rank or standard linear I2L as spike conversion processes.

2.4 LEARNING METHODS FOR SNN

Just like the first and second generation of neural networks, Spiking Neural Networks have learning rules for performing supervised and unsupervised learning.

The term *synaptic plasticity* refers to the process in the brain that adjusts, forms, or removes synapses between neurons. To be more precise, this process alters the synaptic weights and therefore alters the strength of the synaptic connections. The adjustments can result in two types of changes, *potentiation* if the weights are strengthened or *depression* if the weight values are decreased, and the effects can be *long term* or *short term*.

2.4.1 *Unsupervised learning*

Most of the ideas in unsupervised learning take as an inspiration the famous quote also called *Hebbian learning rule* :

> *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.*
>
> — Donald O. Hebb [21]

The general idea is that if any two neurons happen to be active at the same time, they will tend to become 'associated' and the activity in one neuron facilitates activity in the other. This theory is sometimes generalised as: "*Cells that fire together wire together*".

$$W(\Delta t) = \begin{cases} A_P \, e^{\frac{-\Delta t}{\tau_P}} & \Delta t > 0 \\ A_D \, e^{\frac{\Delta t}{\tau_D}} & \Delta t < 0 \end{cases} \tag{17}$$

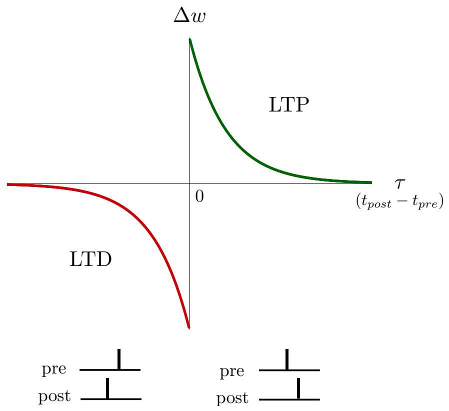


Figure 10: A common STDP window. In green Long Term Potentiation (LTP), in red Long Term Depression (LTD). The height of the windows is determined by parameters A while the width is regulated by the decay parameter τ

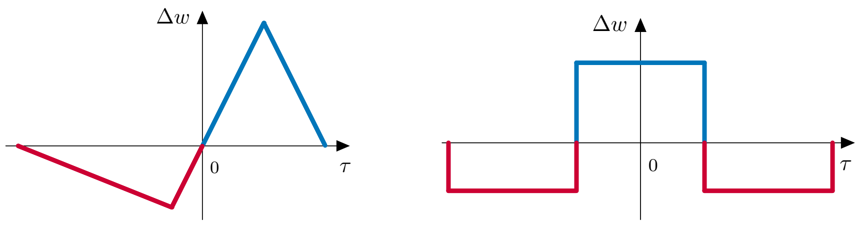
2.4.1.1 STDP

Spike-Timing Dependent Plasticity (STDP) is a temporal Hebbian rule which performs adjustments to the synaptic weights taking into account the *precise time spikes are fired*. This kind of synaptic plasticity shows a maximal effect when the presynaptic neuron fires occurring nearly at the same time of the postsynaptic neuron firing. If the two spikes, pre and post are distant in time, the weight remains unchanged.

Formally, STDP is a function of the presynaptic and postsynaptic spike trains that depends on a function W that ensures the previously cited Hebbian rules are applied:

$$\Delta w_{ji}(t) = \sum_{t_{pre}} \sum_{t_{post}} W(t_j^{post} - t_i^{pre}) \; . \tag{16}$$

A very common function W is the one shown in Figure 10, which potentiates connections where the presynaptic spike happens shorty before the postsynaptic spikes $(t_j^{post} - t_i^{pre}) \simeq 0^+$, while decreases connections if the presynaptic spike happens after the postsynaptic spike $(t_j^{post} - t_i^{pre}) \simeq 0^-$ and the potentiation and depression windows have an exponential behaviour, both decreasing their effect in presence of a bigger time delta between pre/post synaptic spikes Δt. However, this is not the only STDP window possible, there are some other windows that have a linear decrease towards $\pm\infty$, and some others who relax the Hebbian rule and strengthen weights when $(t_j^{post} - t_i^{pre}) \simeq 0$, regardless of the sign. Additional windows can be found in Figure 11. The STDP window in Figure 11b is commonly used for inhibitory connections rather than for excitatory connections.



(a) STDP window with linear increase near zero followed by linear decrease

(b) Symmetric STDP window for inhibitory connections

Figure 11: Some other STDP windows.

Another important aspect when modifying the synaptic ways is not only the function in which the Δw is updated, but also how the update is performed. Updates can be either done with an additive update rule:

$$w \leftarrow w + \Delta w \;, \tag{18}$$

or a multiplicative update rule:

$$w \leftarrow w(1 + \Delta w) \;. \tag{19}$$

STDP with an additive update rule has been shown to generate competition between the synapses [22] and results in a group whose synapses are strongly reinforced and another group whose synapses are are almost completely depressed. However there are also works which support multiplicative STDP claiming it shows pathway specificity [23].

The process with this learning rule is intrinsically not supervised, and it can be successfully used to detect and learn patterns in data in an unsupervised manner [24], even when such patterns are embedded in noise. This means that this learning rule would reduce the need of the highly costly activity of labelling data.

2.4.2 *Supervised learning*

One of the disadvantages of SNN is that the spike generation function is not differentiable and therefore it is not directly compatible with the standard error backpropagation algorithms. Some papers have proposed new backpropagation mechanisms for learning synaptic weights and axonal delays. One of those is SLAYER [25] which is based on a temporal credit assignment policy for backpropagating error to preceding layers. Other approaches like ReSuMe [26], have instead tried alternative ways to achieve supervised learning, in this particular approach, authors extend the Widrow-Hoff rule [1] to SNNs.

1 The Widrow-Hoff rule, also known as the *delta rule*, is a learning rule that attempts to minimise the error in the output of a neural network through gradient descent.

2.5 INFORMATION DECODING

Once the input spikes have been processed with neurons implementing a particular learning rule, we normally want to make sense of the output spikes. In most scenarios, we have multiple neurons which are competing against each other, and the goal is to determine who is the “winner of the competition". To do so there are two main approaches:

- Highest Firing Rate (HFR): as its name states, the winner of the competition is the neuron which has the highest firing rate, and thus has more output spikes than any other neuron.
- First to Fire (F2F): this approach assigns as the winner the first neuron to emit an output spike. Usually this approach is used in networks where neurons are trained to fire first in particular situations and often use a temporal WTA mechanism.

2.5.1 *Lateral Inhibition*

Lateral Inhibition is a mechanism used for to refine selectivity. When a neuron has fired, the neuron prevents that the neighbouring neurons spread their action potential, thereby achieving *inhibition*. This mechanism not only refines selectivity but also leads to competition among neurons.

A related principle is Winner-Take-All (WTA). In this principle, neurons compete against each other to be chosen as a winner, and as a result, the responses of the weaker neurons are suppressed, typically causing them to become inactive. There are also temporal WTA strategies in which the first spike is used to determine the winner (F2F) and causes the suppression of other responses. Because of the inherent competitive effect of lateral inhibition, it can be used to realise a Winner-Take-All mechanism [27].

2.5.2 *Homeostasis*

When using layers of multiple neurons, we normally try to achieve equal firing rates between the neurons in order to prevent that some neurons dominate the response pattern, and neurons differentiate between them. This can be done with an adaptive spiking threshold which is sometimes called *homeostasis*. Instead of having a fixed spiking threshold, we have a fixed resting threshold and then an additive dynamic threshold that increases at every spike fired by the neuron and decays exponentially back to zero with time. The more a neuron fires, the more input it requires to fire again in the near future.

Homeostasis has been used in combination with lateral inhibition, and an additive STDP update rule to perform digit recognition [28].

2.6 NEUROMORPHIC PROCESSORS

To conclude this chapter we take a brief moment to describe neuromorphic hardware for SNNs: neuromorphic processors.

Neuromorphic hardware is a term that encompasses any electrical device — be it analog, digital, or hybrid — which mimics the natural biological structures of the human brain and nervous system.

Modern computing systems are ill-suited for an efficient implementation of SNNs, this is due to their underlying von Neumann architecture which is bottlenecked by the need to constantly exchange data between logic and memory units which are physically separated. A neuromorphic processor is a type of neuromorphic hardware which addresses the limitations of the von Neumann architecture and concretises SNN models in an efficient way, making them useful for practical applications.

Usually neuromorphic processors are categorized according to their neuronal model implementation. We may have software neurons like the SpiNN5 processor from the University of Manchester [29], digital neurons like IBM's TrueNorth [30] or Intel's Loihi [31] processor, or analog neurons like the Braindrop processor from Stanford University [32].

Neuromorphic processors have made SNNs very appealing as the processing of a single spike on these chips may only consume a few pJ or even some fJ (10^{-15}J) of energy [6]. This fact potentially presents the possibility of building neural networks which are energy efficient, and as it has been shown [5], potentially more computationally powerful, and amazingly fast.

Despite the advances in the domain, we are still far away from a achieving something nearly similar to a biological brain. The University of Manchester has built a massively parallel supercomputer based on SNNs. This machine called SpiNNaker (Spiking Neural Network architecture) [29] , occupies about 11 standard 19-inch racks, requires approximately 100kW and is capable of modelling interactions of roughly a billion neurons which represents only one percent of the human brain.

3 EVENT-BASED VISION

3.1 EVENT-BASED CAMERAS

Event-based cameras are a new type of vision sensors whose working principle is biologically inspired. Differently from conventional vision sensors that collect image frames at a fixed rate, these type of sensors asynchronously output changes on pixel-level brightness in the form of events.

To be more precise, this type of sensors emit a tuple called *event* $\text{Event}(x, y, ts, p)$ if at time ts the pixel in coordinate (x, y) suffers a change in log-brightness intensity which is greater than a threshold C called contrast sensitivity, if the change is positive we emit an event with positive polarity (usually represented as ON or with +1), otherwise if the change is negative we emit an event with negative polarity (usually represented as OFF or with -1 or 0).

Formally, the emission of an event e is determined by Equations 20, 21 and it can be easily visualised in Figure 12 :

$$e_i^+ \quad \text{iff} \quad \log I(\mathbf{x}, t_i) - \log I(\mathbf{x}, t_i - \Delta t) \geqslant C\,, \tag{20}$$

$$e_i^- \quad \text{iff} \quad \log I(\mathbf{x}, t_i) - \log I(\mathbf{x}, t_i - \Delta t) \leqslant C\,. \tag{21}$$

One of the main exponents of these kind of sensor is the Dynamic Vision Sensor (DVS) developed by ETH Zurich [33] which tries to imitate three main cell types that are in our eyes (illustrated in Figure 13):

- a **cone cell**, by means of a fast logarithmic photoreceptor that converts current intensity I from photodiodes into a logarithmic voltage.
- a **bipolar cell** by means of a differential circuit which amplifies and generates a negative or positive event.
- a **ganglion cell** by means of a comparator made of two transistors which act as a comparator for ON-OFF events.

The DVS is not the only event-based sensor out there, there have also been other sensors such as the Asynchronous Time-based Image Sensor (ATIS) which operates under the same concept but in a sightly different way.

Due to the fact that both the mechanism and inner structure reflect the human retina, event-based cameras are sometimes also called "*silicon retinas*".

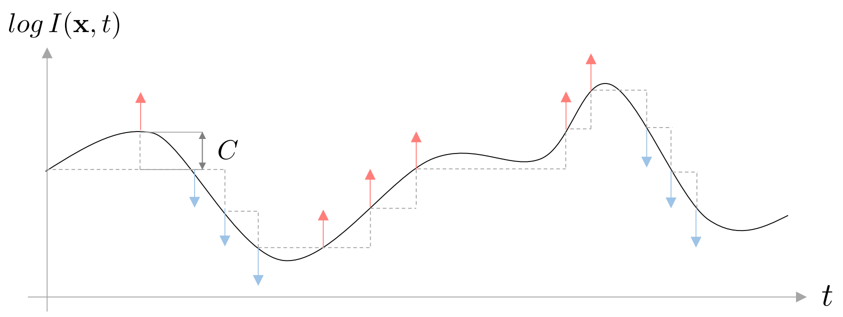


Figure 12: Working principle of an event-based camera. Events are triggered when the intensity increases or decreases beyond the contrast sensitivity C.

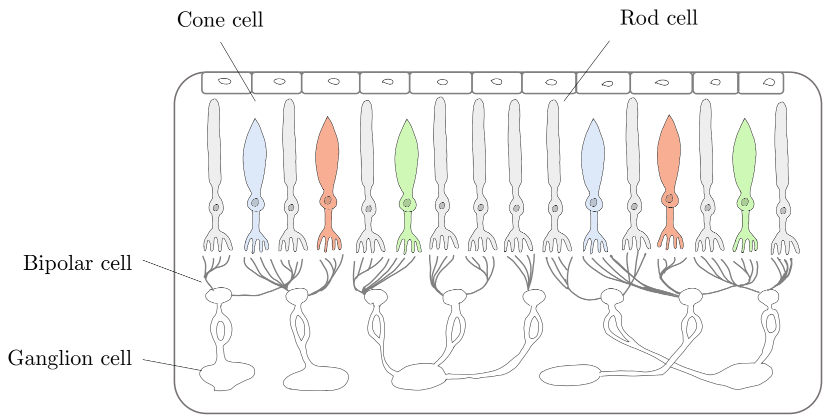


Figure 13: Inner structure of the human retina. *Rod cells* and *Cone cells* are the two type of photoreceptors that can be found in the human retina. Rods have a low spatial acuity while cone cells have a high spatial acuity and respond differently to light of different wavelengths, thus making the latter cell type the one responsible for color vision. Nonetheless, event-based camera sensors do not distinguish wavelengths therefore they do not possess color vision capabilities.

These kind of cameras offer some considerable advantages over standard cameras, namely a very high dynamic range, avoidance of motion blur, and a latency in the microsecond range. Moreover, in non highly dynamic scenes, the visual information gathered is usually very sparse which requires lower transmission bandwidth, storage capacity, processing time, and power consumption compared to conventional imaging sensors. It also has the advantage there is no relevant information lost between frames as relevant information is emitted asynchronously through the form of events.

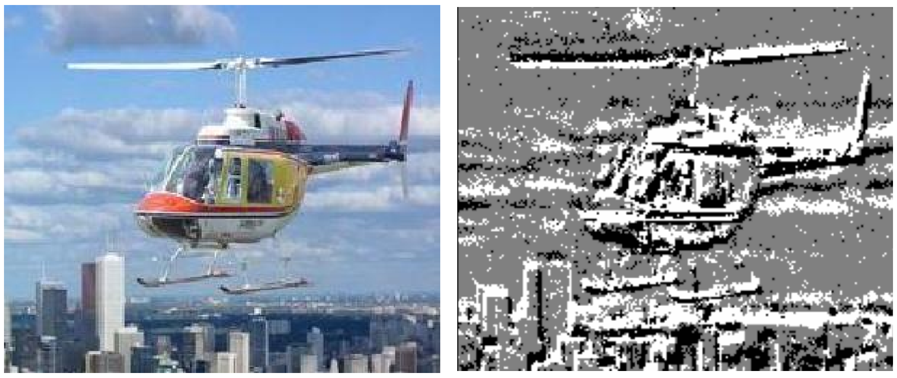


Figure 14: To the left a standard camera image, to the right a grayscale view of an event based image of the same scene with positive events represented as white pixels and negative events represented as black pixels. Another common choice is blue for negative events and red for positive events. Images taken from the Caltech101/N-Caltech101 dataset [34].

	Standard Camera	Event Camera
Power Consumption (mean)	1 W	10 mW
Dynamic range	60 dB	140 dB
Max fps	120-960 fps	~1000000 fps
Data Transfer Rate	32MB/s	~1MB/s

Table 1: Advantages of Event-cameras over traditional camera sensors.

Event-based cameras do not aim to substitute conventional image sensors, it is the properties they offer make them really appealing for a wide range of applications in autonomous vehicles and robotics. In particular, these kind of cameras have found applications on domains like image stabilisation, depth reconstruction, and motion segmentation. In later chapters we will explore two of these domains: object recognition and robot control.

As it was mentioned before, an event camera by no means intends to substitute traditional image sensors, indeed event-cameras are unable to "see" static scenes and are unable to perceive color as shown in Figure 14, for these reasons, researchers developed the Dynamic and Active-pixel Vision Sensor (DAVIS) sensor [35]. This sensor combines the best of two worlds: event-based sensors and conventional sensors, and adds an additional component: an Inertial Movement Unit (IMU), which is a combination of accelerometers, gyroscopes and magnetometers. These kind of sensors can thus trigger events and integrate light intensity on pixels at the same time).

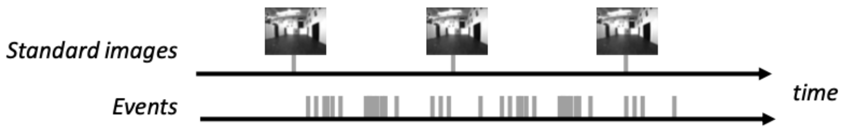


Figure 15: Working mechanism of the DAVIS Sensor.

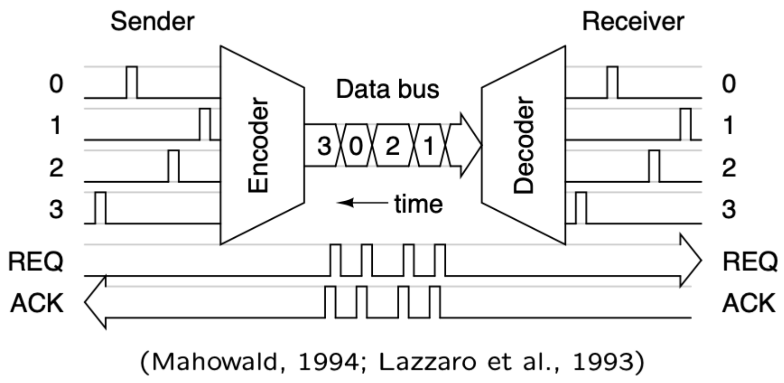


Figure 16: The AER communication protocol.

3.2 ADDRESS EVENT REPRESENTATION

In nature, data transfer from the eye to the brain is done by nearly one million axons of ganglion cells. If we tried to do something similar on event-based vision sensors, we would need to give each pixel its own cable (which unfortunately is unfeasible in current chips). Address Event Representation (AER) is an asynchronous handshaking communication protocol for transferring spikes between bio-inspired chips (such as an event camera and a processing unit like a spiking neural network hardware), that finds a workaround to this problem. This protocol emulates the massive connectivity between cells by time-multiplexing many connections on the same data bus, thus reducing the number of cables needed from N to $\sim \log_2 N$ (Figure 15).

This representation uses streams of events to communicate between chips. These events as described before, are defined as tuples where x and y indicate the pixel reference or location of the event, t indicates the timestamp at which the event took place, and p which indicates the polarity (positive if light intensity has increased, or negative if it has decreased). Each spike is represented by its location (explicitly encoded as an address) and the time the spike occurs (implicitly encoded).

Almost all event-based camera sensors communicate with this protocol, for this reason they are sometimes referred as AER sensors [36, 37, 20].

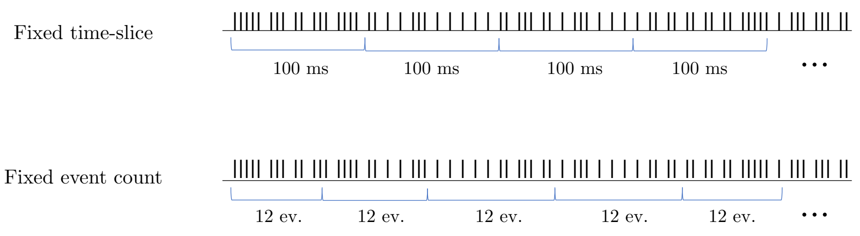


Figure 17: Hard Event Segmentation.

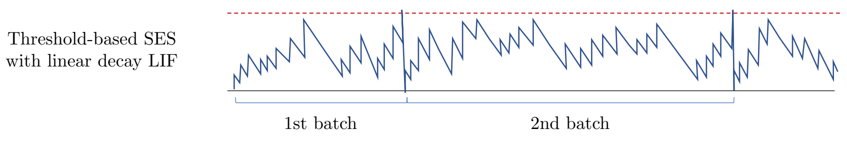


Figure 18: Soft Event Segmentation .

3.3 EVENT SEGMENTATION

The events received from event-based cameras are not always processed as events, sometimes the events are accumulated during a short period of time in order to be able to display an image, or when the event count has reached a certain threshold. This technique shown in Figure 17 goes by the name of Hard Event Segmentation (HES).

A main open-source software used in this domain is the open-source jAER software developed by Delbruck [38]. This software collects events for a certain time slice to create a 2D image of the events. If the time slice is similar to the monitor refresh rate, one can visualise the reconstructed images practically in real time.

Another approach to process events received from AER sensors is Soft Event Segmentation (SES) which groups events in a dynamic way. A common approach, shown in Figure 18 is to use a LIF neuron [1] that increases its potential by a fixed quantity at the arrival of an event and decays with time and when it reaches a threshold it tells the buffer to stop accumulating events, return the current batch and start anew.

One component that realises such approach is called a Motion Symbol Detector (MSD) [39]. The events are accumulated into *segments* by a component called Segment Recorder (SR).

1 Note that the presence of a neuron does not necessarily implies a SES approach, if we use an IF neuron (no decay) with a fixed threshold, this is equivalent to HES with a fixed event count.

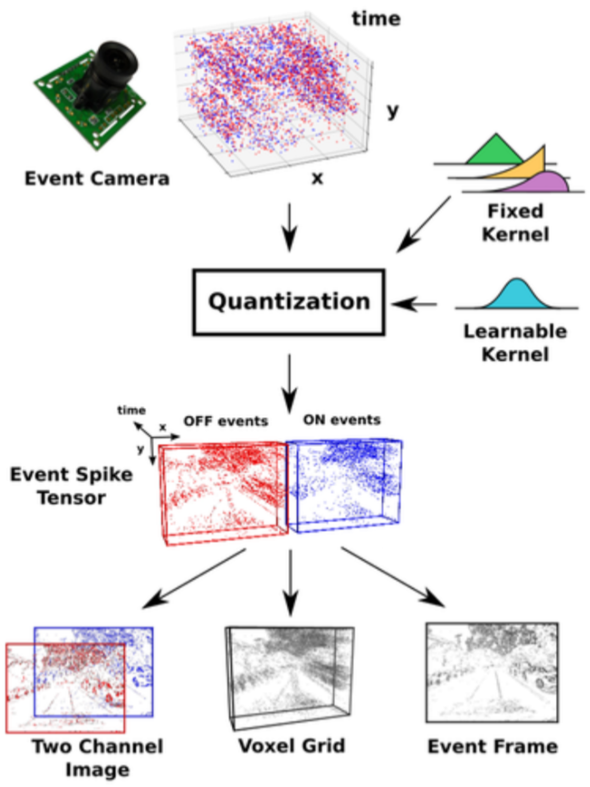


Figure 19: General event-based representation framework. Image taken from [40].

3.4 EVENT REPRESENTATIONS

Once the events have been segmented, events are usually transformed into a useful representation. The most common representations are:

① *event frame*, a 2D histogram where events in the segment are accumulated considering or ignoring polarity.

② *time or leaky surface*, a 2D map where the motion history is taken into account by representing recent motion with brighter intensity values.

③ *voxel grid*, where events are represented as a 3D histogram in a certain time interval (polarity is ignored).

④ *event spike tensor*, a 4D representation which contains a 3D histogram in a certain time interval for each of the polarities.

All this representations are now covered by a general framework developed by Gehrig et al. [40] which converts data into several representations by employing convolutions, quantisation, and projections. This framework that can be observed in Figure 19.

4 REINFORCEMENT LEARNING

4.1 INTRODUCTION

Reinforcement Learning (RL) is one of the three machine learning paradigms which has gained a lot of traction in the last decade. Differently form other paradigms, this area of machine learning is focused on how agents should take actions in an environment in order to maximise a reward. This paradigm results particularly useful when the dynamics of the environment are unknown or are difficult to be modelled, and it has been successfully used to tackle complex problems such as playing the chinese game of Go [1].

In a basic RL problem we have an *agent* or *learner* that interacts with an *environment*. At every time step, the agent selects an action among all the possible actions and the environment responds to that action by presenting to the agent a new situation or *state*, and a value called *reward* that represents the adequateness of the action taken with respect to its goal.

Formally speaking, at each time step t an agent finds itself in a state $S_t \in \mathcal{S}$, where $\mathcal{S}$ indicates the set of all possible states the agent might be in, and executes an action $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(S_t)$ is the set of all available actions in state S_t. As a consequence of the executed action, the agent receives from the environment a reward r_{t+1} and the agent finds itself in a new state S_{t+1}. This is usually visualised with the diagram in Figure 20.

DEFINITION 1 (POLICY): The function that picks the right action at each state is called a *policy* and is represented by π. The policy is what fully defines the behaviour of an agent. Policies can depend on the history of visited states or can depend only on the current state, they can be stationary (time-independent) or non-stationary.

Policies are said to be deterministic if the action taken at each state is completely predictable as it is always the same for the same given state:

$$\pi(s) = a \; . \tag{22}$$

[1] The chinese game of Go represents a great computational challenge as it has been shown that the number of possible states in the game is approximately $2 \cdot 10^{170}$. Such number is vastly greater than the number of atoms in the universe, estimated to be about 10^{80}. In 2015 a computer program based on RL and named *AlphaGo* was able to beat for the first time a human professional Go player.

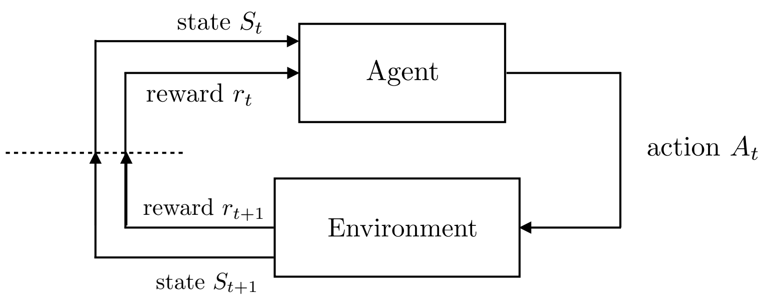


Figure 20: The standard RL paradigm.

Stochastic policies on the other hand, take an action based the probability of selecting that action:

$$\pi(s|a) = \Pr\{A_t = a \,|\, S_t = s\}\ . \tag{23}$$

DEFINITION 2 (MARKOV PROCESS): A stochastic process X_t is said to be a *Markov Process* if it satisfies the Markovian property. Such property involves memorylessness, that is, that the future is independent of the past and it only depends on the present. In other words, the current state captures all the information from history and once the current state is known, history becomes irrelevant:

$$\begin{aligned}&\Pr\{X_{t+1} = j \,|\, X_t = i, X_{t-1} = k, \dots X_0 = k_0\}\\ &=\\ &\Pr\{X_{t+1} = j \,|\, X_t = i\}\ .\end{aligned} \tag{24}$$

DEFINITION 3 (MARKOV REWARD PROCESS): A *Markov Reward process* is a Markov Process that complements the description of a sequence of possible events by adding a reward at each state.

DEFINITION 4 (MARKOV DECISION PROCESS): A Markov Decision Process (MDP) is a Markov Reward Process with decisions. Formally it is defined as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where:

- $\mathcal{S}$ is a set of states
- $\mathcal{A}$ is a set of actions
- P is a state transition probability matrix that states, independently of the reward, how probable it is that we end up in a state s' when taking an action a on a state s.

$$\begin{aligned}P(s' \,|\, s, a) &= \Pr\{S_{t+1} = s' \,|\, S_t = s, A_t = a\}\\ &= \sum_{r \in \mathcal{R}} P(s', r \,|\, s, a)\ .\end{aligned} \tag{25}$$

- R is a reward function that states which is the expected reward when taking an action a on state s. Note that this considers the fact the state transition is not deterministic and therefore we might end up in different states and have different rewards even when taking the same action on the same state. Thus it returns an expected value.

$$\begin{aligned} R(s,a) &= \mathbb{E}[r_{t+1} \,|\, S_t = s, A_t = a] \\ &= \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} P(s', r \,|\, s, a) \ . \end{aligned} \tag{26}$$

- γ is a discount factor $\gamma \in [0,1]$ that regulates the impact of immediate and future rewards.

The advantage of RL is that, by only specifying what we want to achieve and by being able to express a measure of success, we should be able to reach our goal.

HYPOTHESIS 1 (REWARD HYPOTHESIS): A goal can be expressed as the maximisation of the expected cumulative sum of rewards over a given time horizon.

We define the notion of *return* as the sum of all rewards the agent will receive in a given time horizon from the current state, where the time horizon can be infinite, finite, or indefinite if it stops until a criteria is met. The problem with this definition is the fact that the reward might be infinite if the time horizon is infinite. To avoid such problems we introduce the concept of an *infinite horizon discounted return*.

DEFINITION 5 (RETURN): The return G_t is defined as the cumulative (discounted) reward from time-step t:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \ . \tag{27}$$

The value of the parameter γ, also called the *discount factor*, determines how much importance we give to immediate rewards rather than future rewards and model in a way the uncertainty of the future. A value of γ close to 0 is said to be *myopic* as it privileges immediate rewards whereas a value of γ close to 1 is said to be *far-sighted* as it values a lot future rewards. It is however possible to define undiscounted return ($\gamma = 1$) if all sequences terminate.

DEFINITION 6 (STATE-VALUE FUNCTION): Given a policy and an MDP $\mathcal{M}$, we can define what we call the *state-value function* $V_\pi(s)$, that is, the expected return starting from state s if we select actions with policy π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t \,|\, S_t = s] \ . \tag{28}$$

DEFINITION 7 (ACTION-VALUE FUNCTION): Given a policy and an MDP $\mathcal{M}$, we can define what we call the *action-value function* $Q_\pi(s)$, that is, the expected return starting from state s, taking action a and then selecting actions with policy π:

$$Q^\pi(s) = \mathbb{E}_\pi[G_t \,|\, S_t = s, A_t = a] \ . \tag{29}$$

Given these definitions we can reformulate the hypothesis 1 as the maximisation of the value function. We therefore say an MDP is solved when we know the optimal value function. To introduce the notion of optimality we first introduce the notion of ordering.

DEFINITION 8 (POLICY PARTIAL ORDERING): We say that a policy π is better than π' if the value function a state is greater than or equal to that of π' at the same state and the same happens for all states in the MDP.

$$\pi > \pi', \qquad \text{iff} \qquad V^\pi(s) > V^{\pi'}(s) \quad \forall s \in \mathcal{S} \ . \tag{30}$$

DEFINITION 9 (OPTIMAL POLICY): We say that a policy is *optimal* and we represent it as π^* if it is better than or equal to all other possible policies:

$$\pi^* > \pi, \qquad \forall \pi \ . \tag{31}$$

there is also a theorem that states that for any MDP:

- There is at least one deterministic optimal policy π^*.
- All optimal policies achieve the optimal state-value function

$$V^*(s) = \max_\pi V^\pi(s) \ . \tag{32}$$

- All optimal policies achieve the optimal action-value function

$$Q^*(s) = \max_\pi Q^\pi(s, a) \ . \tag{33}$$

As it will be explained in the next sections, an agent generally takes advantage of the existence of an optimal policy and therefore an optimal value function to improve its current policy and eventually learn an optimal policy. Another important thing that is taken advantage of, is the recursiveness of value functions. The equations that explicitly show their recursiveness are known as *Bellman Equations* and are fundamental to find the optimal value function.

4.2 BELLMAN EQUATIONS

DEFINITION 10 (BELLMAN EXPECTATION EQUATION): The Bellman Expectation Equation is a fundamental RL relation that explicitly states the recursivity in the state-value equation. Indeed, the state–value function can be decomposed into an immediate reward plus a discounted value of the successor state:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[r_{t+1} + \gamma V^{\pi}(s') \,|\, S_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\pi}(s') \right) . \end{aligned} \tag{34}$$

If we consider everything in a matrix notation, we can express the Bellman Expectation Equation in a much more compact and simple way:

$$V^{\pi} = R^{\pi} + \gamma P^{\pi} V^{\pi} , \tag{35}$$

where V^{π} and R^{π} are matrices with shape $|\mathcal{S}| \times 1$, and P^{π} is a matrix with shape $|\mathcal{S}| \times |\mathcal{S}|$.By writing the equation this way, it becomes evident that the equation has a direct solution:

$$V^{\pi} = (I - \gamma P^{\pi})^{-1} R^{\pi} , \tag{36}$$

which can be rewritten with the action-value function as follows:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[r_{t+1} + \gamma Q^{\pi}(s',a') \,|\, S_t = s, A_t = a] \\ &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{\pi}(s') . \end{aligned} \tag{37}$$

DEFINITION 11 (BELLMAN OPTIMALITY EQUATION): The Bellman Optimality Equation is another fundamental RL relation :

$$\begin{aligned} V^{*}(s) &= \max_{a} Q^{*}(s,a) \\ &= \max_{a} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{*}(s') \right) . \end{aligned} \tag{38}$$

$$\begin{aligned} Q^{*}(s,a) &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^{*}(s') \\ &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{a'} Q^{*}(s',a') . \end{aligned} \tag{39}$$

As we can see, differently from Equation 34, Equation 38 is non–linear and has no closed form solution. As we will see, in order to compute it there are many iterative solution methods such as Dynamic programming and RL methods like Q-learning or SARSA.

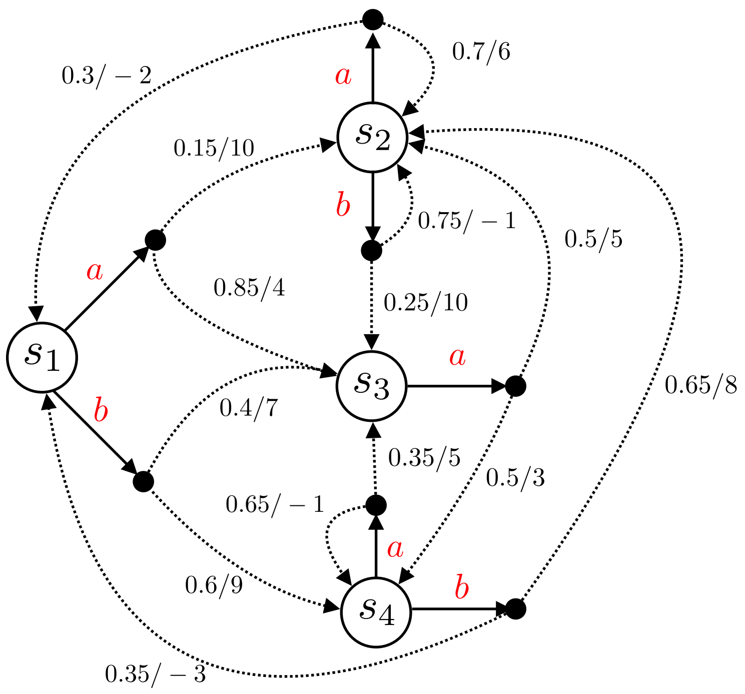


Figure 21: Example of an MDP with 4 states and two actions, transitions are stochastic with *probability/ reward*.

To exemplify the mechanism, let's consider the MDP in Figure 21 and let's assume the policy we follow is to select always action a. For this example we can define the matrices:

$$P^{\pi} = \begin{bmatrix} 0 & 0.15 & 0.85 & 0 \\ 0.3 & 0.70 & 0 & 0 \\ 0 & 0.50 & 0.50 & 0 \\ 0 & 0 & 0.35 & 0.65 \end{bmatrix} \qquad R^{\pi} = \begin{bmatrix} 0.15 \cdot 10 + 0.85 \cdot 4 \\ 0.3 \cdot -2 + 0.7 \cdot 6 \\ 0.5 \cdot 5 + 0.5 \cdot 3 \\ 0.35 \cdot -3 + 0.65 \cdot 8 \end{bmatrix} = \begin{bmatrix} 4.9 \\ 3.6 \\ 4 \\ 4.15 \end{bmatrix} .$$

As a consequence we can write the direct solution to the Bellman expectation equation with $\gamma = 0.6$ as:

$$V^{\pi} = \left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \gamma \begin{bmatrix} 0 & 0.15 & 0.85 & 0 \\ 0.3 & 0.70 & 0 & 0 \\ 0 & 0.50 & 0.50 & 0 \\ 0 & 0 & 0.35 & 0.65 \end{bmatrix} \right)^{-1} \begin{bmatrix} 4.9 \\ 3.6 \\ 4 \\ 4.15 \end{bmatrix} \simeq \begin{bmatrix} 10.76 \\ 9.54 \\ 9.80 \\ 10.18 \end{bmatrix} .$$

Considering the action-value function for state s_1 using Equation 37, we obtain:

$$Q_{\pi}(s_1, a) = 0.15[10 + \gamma V(s_2)] + 0.85[4 + \gamma V(s_3)] = 10.756 \ ,$$
$$Q_{\pi}(s_1, b) = 0.4[7 + \gamma V(s_3)] + 0.6[9 + \gamma V(s_4)] = 14.21 \ .$$

As we can see from the previous example, the value for action b has a higher value than action a. It would then seem logical to change our policy to choose action b in state s_1 instead of choosing action a. This intuitive concept is formalised below.

THEOREM 1 (POLICY IMPROVEMENT): Let π and π' be any pair of deterministic policies — where the action chosen at each state is always the same — such that

$$Q^{\pi}(s, \pi'(s)) \geqslant V^{\pi}(s) \ , \tag{40}$$

then the policy π' must be as good as or better than π, i. e.,

$$V^{\pi'}(s) \geqslant V^{\pi}(s) \ . \tag{41}$$

Enumerating all deterministic policies, evaluating each one of them (computing its V^{π}) and returning the best one is a really naïve approach that is really inefficient as the number of policies is exponential $|A|^{|S|}$. A better approach would be to compute the state–value function V^{π} of a given random policy and improve it greedily by selecting at each state the best action according to the action-value function:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}(s)} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi}(s') \right) \ . \tag{42}$$

After we do a greedy improvement, we then recompute the state-value function until we cannot do any further improvement. Indeed, when we cannot improve the policy, we have $\max_{a \in \mathcal{A}} Q^{\pi}(s, a)$, which is is exactly the bellman optimality equation (Equation 38), meaning that we obtain $V^{\pi} = V^{*}$, and thus the policy π is an optimal policy.

4.3 GENERALIZED POLICY ITERATION

We can thus establish a general method for computing the optimal policy by repeatedly applying two phases:

- *Policy Evaluation*: Estimating V^{π} given a policy π.
- *Policy Improvement*: Generating a better policy $\pi' \geqslant \pi$.

this iterative process called *generalized policy iteration* is visually represented in Figure 22.

We saw before in Equation 36 that it is possible to obtain a direct solution for V^{π}, however, the direct solution is not always feasible as it requires to invert a matrix and such computation might be too complex when the number of states is too high (complexity $O(|\mathcal{S}|^3)$).

To solve this problem we can use other methods for the *policy evaluation* phase, one of such methods is the *Iterative Policy Evaluation*

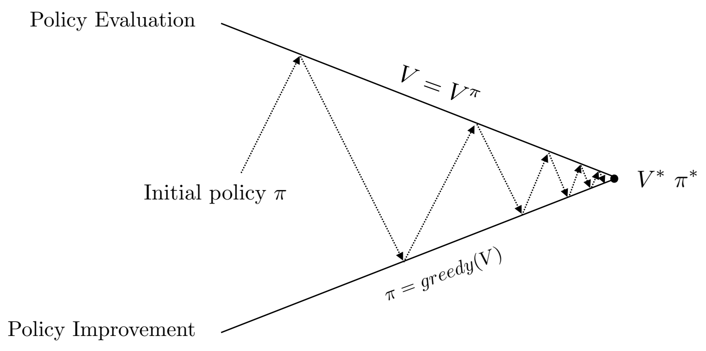


Figure 22: The mechanism of the Generalized Policy Iteration.

which iteratively applies the Bellman expectation equation as an update rule.

Algorithm 1: Iterative policy evaluation

```
Result: $V \simeq V^{\pi}$, the state-value function of policy $\pi$
initialise by setting $V_k(s) \leftarrow 0, \forall s \in \mathcal{S}$
repeat
    $\Delta \leftarrow 0$
    foreach state $s \in \mathcal{S}$ do
        $V_{k+1}(s) \leftarrow$
            $\sum_{a \in \mathcal{A}} \pi(a|s) \big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_k(s')\big)$
        $\Delta \leftarrow \max(\Delta, |V_{k+1}(s) - V_k(s)|)$
        $V_k(s) \leftarrow V_{k+1}(s)$
    end
until $\Delta > \theta$
```

By applying iterative policy evaluation and greedy policy improvement we can obtain the optimal policy, however it is also possible to obtain the optimal policy without the need of two phases. The algorithm that allows to compute the optimal policy is called *value iteration* and applies repeatedly the Bellman Optimality equation:

Algorithm 2: Value iteration

```
Result: $\pi^*$, the optimal policy
initialise by setting $V_k(s) \leftarrow 0, \forall s \in \mathcal{S}$
repeat
    $\Delta \leftarrow 0$
    foreach state $s \in \mathcal{S}$ do
        $V_{k+1}(s) \leftarrow$
            $\sum_{a \in \mathcal{A}} \pi(a|s) \big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_k(s')\big)$
        $\Delta \leftarrow \max(\Delta, |V_{k+1}(s) - V_k(s)|)$
        $V_k(s) \leftarrow V_{k+1}(s)$
    end
until $\Delta > \theta$
```

Differently from Policy Iteration (PI) where we start with a policy and then iteratively improve the policy, the intermediate value functions in Value Iteration (VI) might not correspond to any policy.

There is no method between PI and VI that is clearly better than the other one, while PI is computationally more expensive than VI at each iteration, it typically requires fewer iterations to converge.

4.4 REINFORCEMENT LEARNING METHODS

Until now we have assumed that we have the model of an MDP, meaning that we assumed we had knowledge of MDP transitions/rewards. however, in most cases we do not have any knowledge and we have to learn directly from episodes of experience. In order to do so we need *model-free methods*.

A first method that is model free is the Monte-Carlo (MC) method. This method is quite simple and can be used for policy evaluation (the computation of V^π). It is based on the simple idea of using the empirical mean return instead of the expected return for the state-value function, however, this can be applied only to episodic MDPs meaning that is only applicable to problems that terminate, moreover the algorithm works only with complete episodes of experience.

We can distinguish two variants of the algorithm:

- *every visit*, which averaged returns for every time a state s is visited and leads to a biased but consistent estimator.

Algorithm 3: Every–Visit Monte–Carlo Policy Evaluation

```
Result: $V \simeq V^\pi$, the state-value function of policy $\pi$
initialise by setting $\mathrm{Returns}(s) \leftarrow$ empty list, $\forall s \in \mathcal{S}$
repeat
    generate an episode with policy $\pi$
    foreach state $s \in$ episode do
        foreach occurence of state $s$ in the episode do
            $R \leftarrow$ return following the occurrence of state $s$
            append $R$ to $\mathrm{Returns}(s)$
        end
        $V(s) \leftarrow \mathrm{average}(\mathrm{Returns}(s))$
    end
until desired
```

With this variant it is also possible to update $V(s)$ incrementally with the following update rule:

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t)) \, , \tag{43}$$

where $\alpha = \frac{1}{N(s_t)}$ to track the number of times the state was visited, or a value $\alpha > \frac{1}{N(s_t)}$ to try to forget old episodes, which is particularly useful in non–stationary problems.

- *first visit*, which averages returns only for the first time a state s is visited and leads to an unbiased estimator.

Algorithm 4: First–Visit Monte–Carlo Policy Evaluation

```
Result: $V \simeq V^\pi$, the state-value function of policy $\pi$
initialise by setting $\mathrm{Returns}(s) \leftarrow$ empty list, $\forall s \in \mathcal{S}$
repeat
    generate an episode with policy $\pi$
    foreach state $s$ in the episode do
        $R \leftarrow$ return following the first occurrence of state $s$
        $\mathrm{append}$ $R$ to $\mathrm{Returns}(s)$
        $V(s) \leftarrow \mathrm{average}(\mathrm{Returns}(s))$
    end
until desired
```

To exemplify mechanism, let us suppose we have an MDP with states $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$ (where s_4 is a terminal state), actions $\mathcal{A} = \{a, b, c\}$ and the following episodes where each triplet is (state, action, reward):

$$\begin{aligned}
&(s_1, c, 4) \to (s_3, b, 3) \to (s_2, a, 1) \to (s_4)\ ,\\
&(s_3, a, 5) \to (s_2, b, 1) \to (s_4)\ ,\\
&(s_1, c, 1) \to (s_2, a, 2) \to (s_1, b, 3) \to (s_4)\ .
\end{aligned}$$

we can compute the state-value function as follows:

EVERY-VISIT MC

$$V(s_1) = \frac{8+6+3}{3}, \quad V(s_2) = \frac{1+1+5}{3}, \quad V(s_3) = \frac{4+6}{2}. \tag{44}$$

FIRST-VISIT MC

$$V(s_1) = \frac{8+6}{2}, \quad V(s_2) = \frac{1+1+5}{3}, \quad V(s_3) = \frac{4+6}{2}. \tag{45}$$

Now that we have an algorithm that does a model-free policy evaluation, it seems natural to try and replicate the Generalized Policy Iteration by looking a way to do the model-free policy improvement. It turns out that we can actually perform model-free reedy policy improvement, while greedy policy improvement over $V(s)$ (Equation 42) requires the model of the MDP, greedy policy improvement over $Q(s, a)$ is model–free:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}(s)} Q^\pi(s, a)\,. \tag{46}$$

However there is one major problem with this approach, we cannot use deterministic policies as we also need to explore the different actions available. This is a common problem which goes by the name of *exploration and exploitation trade-off*. If we keep choosing an action greedily without exploring, we would choose what we know to be best according to our current knowledge but we will never choose what may be the actual best action.

To solve this problem we need to use *ε-greedy policy improvement* to ensure continuous exploration.

DEFINITION 12 (ε-GREEDY POLICY IMPROVEMENT): This policy improvement acts greedily with probability $1-\varepsilon$ and chooses an action at random with probability ε:

$$\pi(s,a) = \begin{cases} 1-\varepsilon & \text{if } a_t = \arg\max_{a\in\mathcal{A}(s)} Q^{\pi}(s,a) \\ \dfrac{\epsilon}{|\mathcal{A}|-1} & \text{otherwise} \end{cases} . \tag{47}$$

THEOREM 2: For any ε–greedy policy π, the ε–greedy policy π' with respect to Q^{π}, is an improvement over π, meaning $V^{\pi'} \geqslant V^{\pi}$.

DEFINITION 13 (GREEDY IN THE LIMIT WITH INFINITE EXPLORATION): A learning policy is called Greedy in the Limit with Infinite Exploration (GLIE) if it satisfies the following two properties:

- All state–action pairs are explored infinitely many times

$$\lim_{t\to\infty} N_k(s,a) = \infty\,. \tag{48}$$

- The policy converges on a greedy policy

$$\lim_{t\to\infty} \pi_k(a|s) = \mathbf{1}(a = \arg\max_{a'\in\mathcal{A}(s)} Q^{\pi}(s,a'))\,. \tag{49}$$

DEFINITION 14 (GLIE MONTE-CARLO CONTROL): A policy iteration with policy evaluation of the action-value function $Q(s,a)$ with the following Monte-Carlo every-visit update rule:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \frac{1}{N(s_t,a_t)}(G_t - Q(s_t,a_t))\,, \tag{50}$$

and an ε-greedy policy improvement with

$$\varepsilon \leftarrow {}^{1}/_{k}\,, \tag{51}$$

where k is the episode number, satisfies the GLIE conditions and is called *GLIE Monte-Carlo Control.*

THEOREM 3: GLIE Monte–Carlo Control converges to the optimal action–value function:

$$Q(s,a) \to Q^{*}(s,a)\,. \tag{52}$$

One problem that arises from GLIE Monte-Carlo Control is that even if it is model-free, it cannot be applied to non-episodic tasks (i.e. tasks that do not terminate) like the example in Figure 21, as it can only learn from complete episodes of experience. To solve this problem we can resort to *Temporal Difference Learning*, a different way of learning that can learn from incomplete episodes.

DEFINITION 15 (TEMPORAL DIFFERENCE): Temporal Difference (TD) is a model-free method for policy evaluation that learns directly from episodes of experience even when episodes are incomplete. The learning updates a guess towards a guess instead of using the full unbiased return G_t like MC methods.

The update rule employs a biased estimate of $V^\pi(s_t)$ called *TD target*:

$$\text{TD target} = r_{t+1} + \gamma V(s_{t+1}) \,. \tag{53}$$

This biased estimate has a much lower variance w.r.t. MC as it only depends on one random action, transition, reward instead of many as in the return G_t.

The update rule is thus:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \,. \tag{54}$$

DEFINITION 16 (n-STEP TEMPORAL DIFFERENCE): As we just saw, the main difference between TD and MC is the fact that TD takes a look one step into the future and then uses an estimate accounting all future instants, while MC could practically take a look at infinite steps into the future (obviously it is limited by the length of the episode).

$$\begin{aligned} n = 1 \quad (\text{TD}) \qquad & G_t^{(1)} = r_{t+1} + \gamma V(s_{t+1}) \\ n = 2 \qquad\qquad & G_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2}) \\ \vdots \qquad\qquad & \quad \vdots \\ n = \infty \quad (\text{MC}) \qquad & G_t^{(\infty)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \gamma^{T-1} r_T \end{aligned} \tag{55}$$

The use of n-step Temporal Difference is in a way an approach that tries to find the balance in the “bias-variance" trade-off, by selecting an approach which achieves a good performance on both aspects. Instead of using the unbiased MC approach or the low variance TD, it uses something in-between by using an n-step return defined as:

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}) \,. \tag{56}$$

The update rule is thus:

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t^{(n)} - V(s_t)) \,. \tag{57}$$

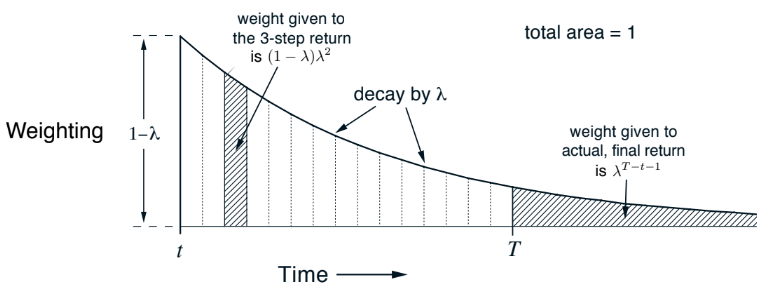


Figure 23: A graph of the G_t^λ weighting taken from Sutton and Barto [41].

Well now that we know there is a model-free method to perform policy evaluation that can use incomplete episodes, we obviously wonder whether it is possible to do a sort of Generalised Policy Iteration like we did with Monte-Carlo. The answer is yes, and such approach is called *SARSA*.

DEFINITION 17: *SARSA* is a control method that uses TD as policy evaluation method for the action-value function $Q(s, a)$ and uses ϵ-greedy policy improvement. The names derives from the fact that all that is needed to perform updates is a state s_t, an action a_t, a reward r_{t+1} and the future state s_{t+1} and future action a_{t+1}:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (58)$$

THEOREM 4: SARSA converges to the optimal action–value function $Q(s, a) \to Q^*(s, a)$ under the following conditions:

- GLIE sequence of policies $\pi_t(s, a)$.
- α_t satisfies the Robbins-Monro conditions:

$$\sum_{t=0}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty \quad . \quad (59)$$

Researchers came with the idea of averaging n–step returns over different n in order to combine information from different time-steps. They came up with what we call λ-return which uses the weighting in Figure 23, expressed by the following equation:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1}\, G_t^{(n)} . \quad (60)$$

If we use the λ-return instead of the n-step return for policy evaluation we have the following update rule:

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t^\lambda - V(s_t)) . \quad (61)$$

This policy evaluation method, which takes the name of *Forward–view* TD(λ), has the problem that it looks infinitely into the future, can only be used in episodic tasks and works only with complete episodes, just like MC. To solve this problem, researchers introduced *Backward–view* $\mathrm{TD}(\lambda)$ which works with incomplete episode sequences and is based on *eligibility traces*.

The *Backward–view* $\mathrm{TD}(\lambda)$ does an update which is in proportion of the TD-error δ_t (i.e. TD-target - $V(s_t)$) and an eligibility trace $e_t(s)$ which combines recency and frequency heuristics to properly assign credit to the state visited given the actual rewards. An example for the eligibility trace $e_t(s)$ is the *accumulating trace*

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{1}(s = s_t) \ , \tag{62}$$

however, it has the problem that frequently visited states can have eligibilities greater than 1 and that can be a problem for convergence. To solve this problem we normally use a *replacing trace* that instead of adding 1 when a state is visited, it simply sets the eligibility to 1:

$$e_t(s) = \begin{cases} \gamma\,\lambda\,e_{t-1}(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases} . \tag{63}$$

Just like the standard TD method, TD(λ) can be used as the policy evaluation phase of the model-free action-value function $Q(s, a)$, which coupled with an ϵ-greedy policy improvement, makes up a control method that converges to the optimal policy. Such method is called SARSA(λ) and works like follows:

Algorithm 5: SARSA(λ)

```
Result: $Q^*$, the optimal action-value function
initialise by setting $Q(s, a)$ arbitrarily and $e(s, a) \leftarrow 0, \forall s, a$
repeat
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using $\varepsilon$-greedy policy
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $e(s, a) \leftarrow e(s, a) + 1$
    foreach state-action pair $s, a$ do
        $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
        $e(s, a) \leftarrow \gamma\lambda e(s, a)$
    end
until $s$ is terminal
```

4.5 OFF-POLICY LEARNING

Until now we have considered on-policy learning methods, meaning that we learn from a policy only by trying it. *Off-policy* learning is a group of methods which learn about other policies without necessarily trying them explicitly, just like humans learn from observation of other agents. This means that we can learn about multiple policies while following only one, and we can even learn about the optimal policy while following exploratory policy.

DEFINITION 18: Importance sampling is an off-policy technique to estimate the expectation $\mathbb{E}[f(x)]$ of a different distribution w.r.t. the distribution used to draw samples. Assuming we have two distributions $P(x)$ and $Q(x)$:

$$\mathbb{E}_{x \sim P}[f(x)] = \mathbb{E}_{x \sim Q}\left[\frac{Q(x)}{P(x)} f(x)\right] . \tag{64}$$

Importance sampling can be applied to Monte Carlo Methods as well as SARSA methods.

4.5.1 *Monte Carlo Importance Sampling*

In MC methods we apply importance sampling corrections along the whole episode according to the similarity between $\bar{\pi}$, the policy which generated the sample rewards, and π, the policy to evaluate:

$$G_t^{\mu} = \frac{\pi(a_t|s_t)}{\bar{\pi}(a_t|s_t)} \frac{\pi(a_{t+1}|s_{t+1})}{\bar{\pi}(a_{t+1}|s_{t+1})} \cdots \frac{\pi(a_T|s_T)}{\bar{\pi}(a_T|s_T)} G_t . \tag{65}$$

On the update rule, we use this corrected return:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(G_t^{\mu} - Q(s_t, a_t)) . \tag{66}$$

Obviously this importance sampling can dramatically increase variance.

4.5.2 SARSA *Importance Sampling*

On the SARSA method we use a similar thing, but we weight the TD target $r + \gamma Q(s', a')$ according to similarity between policies.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \frac{\pi(a_t|s_t)}{\bar{\pi}(a_t|s_t)} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) . \tag{67}$$

Since this update rule has a single correction, errors do not compound themselves like in the MC case, achieving a much lower variance.

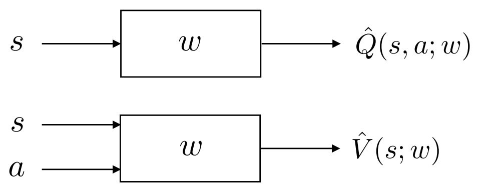


Figure 24: Value function approximation, w represents some parameters of the function which are used for the computation of the output.

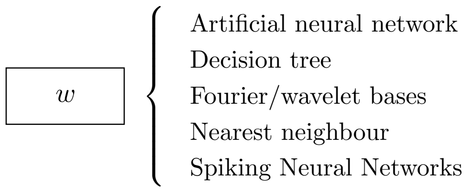


Figure 25: Some example of the functions that can be used for value function approximation.

4.5.3 *Q-learning*

Q-learning is an off-policy method capable of learning the optimal policy π^* by using the behaviour policy $\bar{\pi}$. To do so the algorithm has the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)). \tag{68}$$

This algorithm converges to the optimal policy if we use an ε-greedy strategy.

4.6 VALUE FUNCTION APPROXIMATION

So far we have implicitly assumed that the MDP is not really big and admits a *tabular representation*, that is, that we can keep all the state-action value function in a vector or in a matrix. Unfortunately, many problems have enormous or even continuous state or action spaces which makes the current approaches useless as we cannot keep track of so many values. Moreover as the state spaces may be enormous we may end up quite frequently in a lot of cases we have never seen before, so it is important to find a way to generalise and take good decisions even in unseen situations.

Value Function Approximation (VFA) is a technique that aims to solve the previously listed problems by using a parametrised function to represent the state or the action-value function of the MDP. Let us suppose for a moment that we have an oracle which is able to tell us the true value for $V^\pi(s)$, our goal then becomes to find the parameter vector $\mathbf{w}$ for our function approximator that minimises the loss between the true value $V^\pi(s)$ and its approximation $\hat{V}^\pi(s;\mathbf{w})$. To do so, we can use the mean-squared error as loss function:

$$J(\mathbf{w}) = \mathbb{E}_\pi[(V^\pi(s) - \hat{V}^\pi(s;\mathbf{w}))^2]\,, \tag{69}$$

and try to perform gradient descent to find a local optimum:

$$\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_\mathbf{w} J(\mathbf{w})\,. \tag{70}$$

In reality we obviously do not have an oracle which tells us the true value, so we obviously have to resort to model-free methods. Moreover, the MC and TD estimates for $V^\pi(s)/Q^\pi(s,a)$ update steps that we saw in Section 4.4, must now include the fitting of a function approximator that does the value function approximation.

4.6.1 *Linear Value Function Approximation*

Linear Value Function Approximation is a way of doing VFA, this particular class of VFA represents a value function for a particular policy π using a weighted linear combination of features:

$$\hat{V}^\pi(s;\mathbf{w}) = \mathbf{x}(s)^T\,\mathbf{w}\,, \tag{71}$$

where $\mathbf{x}(s)$ is a feature vector used to represent state s. We can try to perform gradient descent to find a local optimum:

$$\Delta\mathbf{w} = \underbrace{-\frac{1}{2}\alpha}_{\text{step-size}}\;2\,\underbrace{(V^\pi(s) - \mathbf{x}(s)^T\,\mathbf{w})}_{\text{prediction error}}\;\underbrace{\mathbf{x}(s)}_{\text{feature value}}\;, \tag{72}$$

note that this still assumes the presence of an oracle as we have the true value $V^\pi(s)$ in the prediction error.

4.6.2 *Monte-Carlo Value Function Approximation*

Monte-Carlo Value Function Approximation is essentially doing supervised learning on a state of (state,return) pairs $\langle s_1, G_1\rangle$, $\langle s_2, G_2\rangle$, $\langle s_3, G_3\rangle$. It thus substitutes the true value $V^\pi(s)$ for the full expected return G_t, similar to what we did in the standard MC policy evaluation approach (Equation 43).

$$\Delta\mathbf{w} = \alpha(G_t - \hat{V}^\pi(s;\mathbf{w}))\nabla_\mathbf{w}\hat{V}^\pi(s;\mathbf{w})\,. \tag{73}$$

THEOREM 5: MC with VFA converges to the weights which have the minimum mean squared error possible.

Actually if we have a set of episodes from a policy π we can analytically solve for the best linear approximation that minimises the error on the dataset as if it were a simple regression problem

$$\mathbf{w} = (X^TX)^{-1}X^T\mathbf{G}\,. \tag{74}$$

where X is a matrix of the features of each of the states $\mathbf{x}(s_i)$ and $\mathbf{G}$ is a vector of all returns.

4.6.3 *Temporal-Difference Value Function Approximation*

Just like in tabular representation methods, we cannot use MC to do online learning as we need complete episodes. To solve this we recur to Temporal-Difference Value Function Approximation which essentially does supervised learning on a state of (state, TD return) pairs $\langle s_1, r_1 + \gamma \hat{V}^{\pi}(s_2; \mathbf{w})\rangle$, $\langle s_2, r_2 + \gamma \hat{V}^{\pi}(s_3; \mathbf{w})\rangle$, ... It thus substitutes the true value $V^{\pi}(s)$ for the TD return $r_t + \gamma \hat{V}^{\pi}(s_{t+1}; \mathbf{w})$, similar to what we did in the standard TD policy evaluation approach :

$$\Delta \mathbf{w} = \alpha \left(r + \gamma \hat{V}^{\pi}(s'; \mathbf{w}) - \hat{V}^{\pi}(s; \mathbf{w})\right) \nabla_{\mathbf{w}} \hat{V}^{\pi}(s; \mathbf{w}) \,. \tag{75}$$

theorem 6: TD with VFA converges to weights which are within a constant factor of the minimum mean squared error possible.

4.6.4 *Control using Value Function Approximation*

We would also like to find the optimal policy in cases in which the state-actions are too big to use tabular representation. As we saw, we can to policy evaluation with Value Function Approximation. Can we do some sort of Generalised Policy Iteration also in this case? Well the answer is yes, but we need to do the same things we did on the model-free control approaches, we need to operate on Q instead of V and perform ε-greedy policy improvement.

Monte-Carlo methods

$$\Delta \mathbf{w} = \alpha \left(G_t - \hat{Q}^{\pi}(s, a; \mathbf{w})\right) \nabla_{\mathbf{w}} \hat{Q}^{\pi}(s, a; \mathbf{w}) \ . \tag{76}$$

SARSA methods

$$\Delta \mathbf{w} = \alpha \left(r + \gamma \hat{Q}^{\pi}(s', a'; \mathbf{w}) - \hat{Q}^{\pi}(s, a; \mathbf{w})\right) \nabla_{\mathbf{w}} \hat{Q}^{\pi}(s, a; \mathbf{w}) \ . \tag{77}$$

Q-learning methods

$$\Delta \mathbf{w} = \alpha \left(r + \gamma \max_{a'} \hat{Q}^{\pi}(s', a'; \mathbf{w}) - \hat{Q}^{\pi}(s, a; \mathbf{w})\right) \nabla_{\mathbf{w}} \hat{Q}^{\pi}(s, a; \mathbf{w}) \ . \tag{78}$$

4.6.5 *Deep Q-Networks*

Linear VFA works pretty well if it is given a right set of feature representation for the states $\mathbf{x}(s_i)$, however it is often hard to design that set by hand. Deep neural networks have been very popular and useful because they do not require an explicit specification of features. We can thus think of Deep Neural Networks as a value function approximator which is a composition of multiple functions (linear and non-linear transformations) and that can consequently represent a really complex space of functions. Neural networks that compute the action-value function Q are called Deep Q-Networks (DQNs).

4.6.5.1 *Deep Q-Learning*

Q-learning is a model-free technique that is *off-policy* that is very useful in tabular representation methods, however, it has the problem that if we use VFA instead, it can fail to converge due to the correlation between the samples and the non-stationary targets (target changes at each step). Deep Q-Learning is a method that tries to apply the same idea of Q-learning but addresses the problems that lead to the possible divergence of the method. To do so it uses:

- Experience Replay: a mechanism that removes correlations in samples by using a random sample (s, a, r, s') from a *replay buffer* storing prior experience, instead of using the most recent sample. This also allows to use each sample more than once and to update the Q-function more than once at different steps.
- Fixed Q-targets: to improve stability and avoid weights that go to infinity, we fix the target weights used in the target part of the update for multiple updates. Therefore when computing the update we will have two kind of weights: a set of weights $\mathbf{w}^-$ used in the target that changes occasionally, and a set of weights $\mathbf{w}$ which are the ones being updated.

Algorithm 6: Deep Q-learning with Experience Replay

```
Result: $Q^*$, the optimal action-value function
initialise replay buffer $\mathcal{D}$ to initial capacity $N$
initialise $\hat{Q}^\pi(s, a; \mathbf{w})$ function with random weights
foreach episode $i$ do
    using $\varepsilon$-greedy select an action and observe transitions
    add the transition $(s, a, r, s')$ to $\mathcal{D}$
    sample a (minibatch) random transition from $\mathcal{D}$
    compute updates: $\Delta\mathbf{w} =$
      $\alpha\left(r + \gamma \max_{a'} \hat{Q}^\pi(s', a'; \mathbf{w}^-) - \hat{Q}^\pi(s, a; \mathbf{w})\right) \nabla_{\mathbf{w}} \hat{Q}^\pi(s, a; \mathbf{w})$
    if $i \mod k == 0$ then $\mathbf{w}^- \leftarrow \mathbf{w}$
end
```

4.6.6 *Double DQN*

The Q-Learning algorithm has a maximisation bias as it takes the maximum over an estimated Q value function. In Double Q-Learning we can maintain two separate value functions in which we use the value of one as the target for the other to reduce this bias and get a significantly increase in performance.

Algorithm 7: Double Q-learning

```
initialise $Q_1(s,a)$ and $Q_2(s,a)$, $\forall s \in \mathcal{S}, a \in \mathcal{A}$
foreach episode do
    select action $a_t$ using $\epsilon$-greedy with
      $\pi(s) = \arg\max_a Q_1(s_t,a) + Q_2(s_t,a)$
    Observe reward $r_t$ and next state $s_{t+1}$
    choose randomly between $\{\mathtt{update}(Q_1), \mathtt{update}(Q_2)\}$
    if $\mathtt{update}(Q_1)$ then
        $Q_1(s_t,a_t) \leftarrow$
          $Q_1(s_t,a_t) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} Q_2(s_{t+1},a') - Q_1(s_t,a_t))$
    else
        $Q_2(s_t,a_t) \leftarrow$
          $Q_2(s_t,a_t) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} Q_1(s_{t+1},a') - Q_2(s_t,a_t))$
    end
end
```

If we combine this idea with the previous Deep Q-learning Network (DQN) we end up with a method that we call Double DQN.

$$\Delta \mathbf{w} = \alpha \left(r + \gamma \hat{Q} \left(\arg\max_{a'} \hat{Q}^{\pi}(s',a';\mathbf{w}); \mathbf{w}^{-} \right) - \hat{Q}^{\pi}(s,a;\mathbf{w}) \right) \tag{79}$$

4.6.7 *Prioritized Replay*

The order in which we replay updates is really important and can potentially lead to an exponential improvement in convergence. What the Prioritized Replay method tries to do is to proritise the experience tuple (s,a,r,s') proportionally to the DQN error defined as follows:

$$p_i = \left| r + \gamma \max_{a'} \hat{Q}^{\pi}(s_{i+1},a';\mathbf{w}^{-}) - \hat{Q}^{\pi}(s_i,a_i;\mathbf{w}) \right| . \tag{80}$$

Stochastic Prioritization

$$P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}} . \tag{81}$$

4.6.8 *Dueling DQN*

Instead of using a neural network to compute directly the action value function $Q^{\pi}(s,a)$, this approach computes $V^{\pi}(s)$, and the *advantage function* defined as:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) . \tag{82}$$

The advantage function corresponds to the *regret*, or reward lost for not taking the best action. Using this function can stabilise VFA.

$$\hat{Q}^{\pi}(s,a;\mathbf{w}) = \hat{V}^{\pi}(s;\mathbf{w}) + \left(\hat{A}(s,a;\mathbf{w}) - \frac{1}{|\mathcal{A}|} \sum_{a'} \hat{A}(s,a';\mathbf{w}) \right) . \tag{83}$$

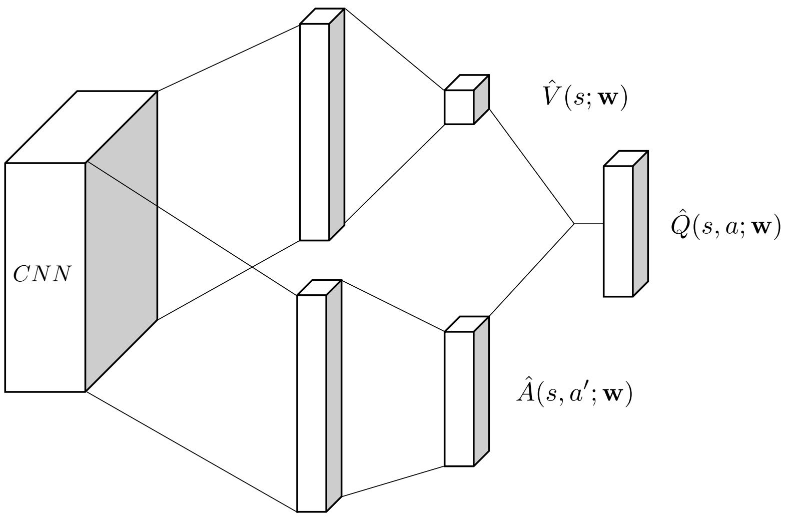


Figure 26: Dueling DQN Architecture.

The dueling DQN architecture can be observed in Figure 26.

4.7 POLICY GRADIENT

So far we have considered *action-value methods* — i.e. methods select which actions to take based on their estimates on action-values pairs (e.g. ε-greedy policy improvement) — however, there are methods that can learn and improve a policy without consulting a state- or action-value function and can optimise the policy directly. Such methods are called *policy gradient methods* and what they do instead, is to learn a parametrised policy:

$$\pi_\theta(a \mid s) = \Pr(a \mid s;\, \theta)\,. \tag{84}$$

These methods are named this way because they usually try to maximise some performance measure $J(\theta)$ and thus their updates approximate a gradient ascent. Note that this does not mean that in these methods the value function is not needed, it may still be used to learn the policy parameter θ, however, the fundamental difference is that it is not required for action selection. Generally speaking, these approaches have better convergence properties and are really effective in very big (or even continuous) action spaces.

Up until now we have considered only deterministic policies (i.e. we always take the same action in the same state) mainly because we are guaranteed that for any MDP there is at least one deterministic optimal policy, however, with policy-based RL methods we can learn stochastic policies, which is really useful in adversarial non-stationary settings or partially observable environments.

Our goal is then to find the best parameters θ that help us achieve the best policy. While in value-function methods we could compare polices through their value functions, here we can use:

- the start value of the policy for episodic environments:

$$J_1(\theta) = V^{\pi_\theta}(s_1)\,. \tag{85}$$

- the average value of the policy for continuing environments:

$$J_{avg(V)}(\theta) = \sum_s d^{\pi_\theta}(s)\, V^{\pi_\theta}(s)\,, \tag{86}$$

$$J_{avg(Q)}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(a\,|\,s)\, Q^{\pi_\theta}(s,a)\,, \tag{87}$$

  where $d^{\pi_\theta}(s)$ is the stationary distribution of the Markov Chain for π_θ.

- the average reward per time-step:

$$J_{avg(R)}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(a\,|,\,s)\, R(s,a)\,. \tag{88}$$

Policy gradient algorithms look for a local maximum in $V(\theta) = V^{\pi_\theta}$ by ascending the gradient of the policy

$$\Delta\theta = \alpha \nabla_\theta J(\theta)\,, \tag{89}$$

where $\nabla_\theta J(\theta)$ takes the name of *policy gradient*.

The estimation of the gradient can be done with two methods:

① *Finite Difference Approximation* which can be applied to the gradient even when the policy is not differentiable. It consists in estimating the k-th partial derivative of the objective function by slightly perturbing in the k-th dimension:

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}\,. \tag{90}$$

② *Policy Gradient Theorem* when we have a differentiable policy.

THEOREM 7 (POLICY GRADIENT THEOREM): An unbiased but noisy estimate of the policy gradient is:

$$\begin{aligned}\nabla_\theta J(\theta) &\simeq \sum_s d^{\pi_\theta}(s) \sum_a Q^{\pi_\theta}(s,a)\, \nabla_\theta \pi_\theta(a\,|\,s) \\ &= \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(a\,|\,s)\, Q^{\pi_\theta}(s,a)\, \frac{\nabla_\theta \pi_\theta(a\,|\,s)}{\pi_\theta(a\,|\,s)} \\ &= \mathbb{E}_\pi\left[Q^{\pi_\theta}(s,a) \nabla_\theta \ln \pi_\theta(a\,|\,s)\right]\,.\end{aligned} \tag{91}$$

4.7.1 REINFORCE

The REINFORCE algorithm is a policy-gradient algorithm that estimates $Q^{\pi_\theta}(s, a)$ with the unbiased MC return G_t. As a consequence, it can only be applied to episodic tasks.

Algorithm 8: REINFORCE, Monte-Carlo Policy-Gradient Method

```
Result: $\pi_\theta \approx \pi^*$
Initialise policy parameter $\theta$ arbitrarily
foreach episode do
    foreach timestep $t = 1$ to $T - 1$ do
        $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a \mid s) G_t$
    end
end
```

4.7.2 *Actor-Critic*

Actor critics are algorithms that learn both a policy and value functions. They are called this way because there are two main components: the *actor* which is the component that learns a parametrised policy $\pi_\theta(a \mid s)$, and the *critic* which is the component that learns about the policy that is currently being followed by the actor in order to 'criticise' the choices it performs. The critic learns a value function with a TD algorithm and such value function allows the critic to compute TD errors which can be used to criticise the actor:

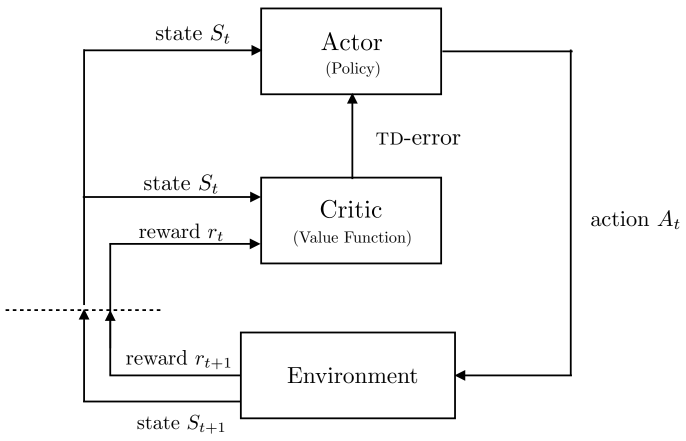
$$\text{TD error} \quad \delta = \left(r + \gamma - \hat{Q}^{\pi_\theta}(s, a; \mathbf{w})\right) , \tag{92}$$

$$\text{critique} \begin{cases} \delta > 0, & \text{action was good, better than estimate} \\ \delta < 0, & \text{action was bad, worse than estimate} \end{cases} . \tag{93}$$

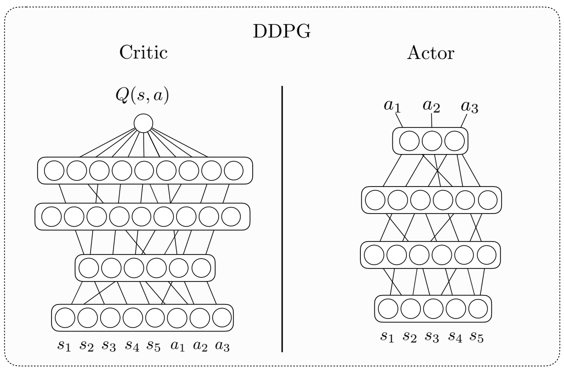
These critiques performed by the critic help the actor to continually update and improve its policy.

Algorithm 9: Actor-Critic Algorithm

```
Result: $\pi_\theta \approx \pi^*$
Initialise policy parameter $\theta$ arbitrarily
Initialise value function parameters $\mathbf{w}$ arbitrarily
foreach episode do
    foreach timestep $t = 1$ to $T - 1$ do
        update policy parameters:
          $\theta \leftarrow \theta + \alpha_\theta \hat{Q}^{\pi_\theta}(s, a; \mathbf{w}) \nabla_\theta \log \pi_\theta(a \mid s)$
        compute TD error $\delta_t$
        update value function parameters:
          $\mathbf{w} \leftarrow \mathbf{w} + \alpha_\mathbf{w} \, \delta_t \nabla_\mathbf{w} \hat{Q}^{\pi_\theta}(s, a; \mathbf{w})$
    end
end
```



(a) Actor Critic Architecture.


DDPG
Critic
Actor
$Q(s, a)$
a_1 a_2 a_3
s_1 s_2 s_3 s_4 s_5 a_1 a_2 a_3
s_1 s_2 s_3 s_4 s_5

(b) DDPG Architecture.

Figure 27: General Actor-Critic vs DDPG architecture.

4.7.3 *Deep Deterministic Policy Gradient*

Deep Deterministic Policy Gradient (DDPG) is an off-policy algorithm used in environments with continuous action spaces that combines some ideas from DQNs (like Experience Replay) and ideas from Actor-Critic architectures.

When we deal with continuous action spaces, we cannot apply DQN. In such cases we need one output neuron for each action and thus we would need an infinite number of action neurons. DDPG solves this by taking as input not only the state but also the action in a *Critic network* that computes the value function. This network is paired to another network, an *Actor network* that selects an action given the state of the environment. Instead of taking the gradient over the weights, we take the gradient over the actions in the Q-function: $\nabla_a \hat{Q}(s, a; \mathbf{w})$. This represents the simple intuition that if we end change the action and we end up with a higher Q value, we should change the action on the actor. The actor should then optimise its policy parameters.

Algorithm 10: DDPG

```
Result: $\pi_\theta \approx \pi^*$
Randomly initialise critic network Q and actor $\mu$ with weights
  $\theta^Q$ and $\theta^\mu$
Initialise target network $Q'$ and $\mu'$ with $\theta^{Q'} \leftarrow \theta^Q$ and
  $\theta^{\mu'} \leftarrow \theta^\mu$
foreach episode do
    Initialise random noise process $\mathcal{N}$ for exploration
    foreach timestep $t = 1$ to $T - 1$ do
        select action $a_t = \mu\{s_t|\theta^\mu\} + \mathcal{N}_t$
        execute action $a_t$ and observe $s_{t+1}$ and reward $r_t$
        store transition $(s_t, a_t, r_t, s_{t+1})$ in the replay buffer
        sample a mini-batch of K transitions from the replay
          buffer
        set $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right)$
        Update critic by minimizing the loss:
        $L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i \mid \theta^Q\right)\right)^2$
        Update the actor policy using the sampled policy
          gradient:
          $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q\left(s, a \mid \theta^Q\right)\Big|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu\left(s \mid \theta^\mu\right)\Big|_{s_i}$

        Update the target networks:
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
        Update policy parameters:
          $\theta \leftarrow \theta + \alpha_\theta \hat{Q}^{\pi_\theta}(s, a; \mathbf{w}) \nabla_\theta \log \pi_\theta(a \mid s)$
        Compute TD error $\delta_t$
        Update value function parameters:
          $\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}} \delta_t \nabla_{\mathbf{w}} \hat{Q}^{\pi_\theta}(s, a; \mathbf{w})$
    end
end
```

4.7.4 *Trust Region Proximal Optimization*

Trust Region Policy Optimization (TRPO) is an on-policy algorithm that can be used for environments with either discrete or continuous action spaces. While normal policy gradient algorithms produce a new policy close in parameter space, this does not guarantee in any way a potential increase in performance, actually the new policy might be substantially worse. TRPO is an algorithm that monotonically increases performance of a policy by choosing the policy with the best increase in performance that is not so different to the old policy. The performance of a policy π_θ w.r.t the old policy $\pi_{\theta_{old}}$ can be measured with the *surrogate advantage*:

$$\mathbb{E}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} A^{\pi_\theta}(s_t, a_t) \right] , \tag{94}$$

where A is the *advantage function*.

While the constraint between consecutive policies is expressed in terms of KL-Divergence, a measure of difference between probability distributions, which determines a "trust region" around the current policy.

Therefore, instead of using a standard gradient update, TRPO uses a natural gradient update which maximises the performance but subject to a KL-divergence constraint between consecutive policies that guarantees that they will not be "too different":

$$\max_\theta \quad \mathbb{E}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} A^{\pi_\theta}(s_t, a_t) \right] , \tag{95a}$$

$$\text{subject to} \quad \mathbb{E}_t \left[\mathrm{KL}(\pi_{\theta_{old}}, \pi_\theta) \right] \leqslant \delta \, . \tag{95b}$$

In practice this algorithm is more stable than DDPG and performs well in practice, but it is less sample-efficient.

4.7.5 *Proximal Policy Optimization*

Given that TRPO is relatively complicated due to the KL constraint, we can think of using a soft constraint instead of a hard one:

$$\max_\theta \; \mathbb{E}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} A^{\pi_\theta}(s_t, a_t) \right] - \beta \mathbb{E}_t [\mathrm{KL}(\pi_{\theta_{old}}, \pi_\theta)] , \tag{96}$$

or we can use a clipped surrogate objective while retaining similar performance:

$$\mathbb{E}[\min(r(\theta)\hat{A}_{\theta_{old}}(s, a), \mathrm{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{old}}(s, a))] \, . \tag{97}$$

this variants of TRPO are called Proximal Policy Optimization (PPO) algorithms.

These algorithms generally are simpler to implement and have a better performance than TRPO.

4.7.6 *Soft Actor Critic*

The Soft Actor Critic (SAC) algorithm is an off-policy Actor-Critic algorithm with a stochastic actor that takes into account the entropy measure of the policy into the reward in order to encourage exploration. Normally this influence is regulated by a parameter α, also known as *temperature parameter*.

This methods uses two value function networks, one computing the state-value function V and other computing the action-value function Q.

The update on the state-value function is given by:

$$\nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)\big(V_\psi(s_t) - Q(s_t, a_t; \mathbf{w}) + \log \pi_\theta(a_t|s_t)\big) , \quad (98)$$

while the update on the action value function is given by:

$$\nabla_\mathbf{w} J_Q(\mathbf{w}) = \nabla_\mathbf{w} Q(s_t, a_t; \mathbf{w})\big(Q(s_t, a_t; \mathbf{w}) - r_{t+1} + \gamma V_{\bar{\psi}}(s_{t+1})\big) , \quad (99)$$

while the policy is updated in a way that minimises the KL-divergence.

4.8 imitation learning

One of the problems with the seen RL methods is their efficiency. Many times an MDP will require a large number of samples to learn a good policy, and sometimes this number is so big that is generally infeasible. So learning policies guided by rewards is a good way of supervision, however it has high sample complexity and is not particularly useful when it is very expensive or not tolerable to fail, or when data is not cheap and is not easy to parallelize. One of the benefits of such methods is that if we give the agent a lot of rewards we an shape behaviour pretty quickly, however, the problem is that manually designing the rewards is not very easy and the specified reward may not be very robust.

An idea that popped out was to use the structure and additional knowledge to help constrain and speed reinforcement learning. This can be done through an *expert* who provides a set o demonstration trajectories (sequences of state-actions) for the desired behaviour, this way the rewards are implicitly specified through demonstrations. Note that this approach is of course only useful if specifying the desired policy or the reward is not easily achievable.

definition 19 (imitation learning problem): In an imitation problem we have the following elements:

- A state space $\mathcal{S}$.
- An action space $\mathcal{A}$.
- A transition model $P(s' \mid s, a)$.
- A set of demonstrations from a teacher policy π^*: $(s_0, a_0, s_1, \dots)$.

From this problem there have been a lot of approaches. *Behavioural Cloning* tries to learn directly the teacher's policy using supervised learning, *Inverse RL* tries to recover the reward function $R(s, a)$ that was not specified. *Apprenticeship learning via Inverse RL* tries to use the recovered reward function R to generate a good policy.

4.8.1 *Behavioural Cloning*

As we briefly mentioned before, *Behavioural Cloning* uses an approach that treats the problem as a standard supervised learning problem. To do so, what we do first is to fix a policy class that will try to learn the mapping $s \to a$ (e.g. neural network, decision tree, etc.) and then we just try to estimate the policy from the training data (i.e. our expert's state-action pairs derived from its demonstration trajectories). However, one of the challenging aspects of this approach is the fact that errors can compound a lot, mainly because the distribution of states that we get depends on the actions taken, therefore a badly selected action can take us into a state that we have never seen before and we do not know how to act on that situation as we are imitating and not generalising.

$$\mathbb{E}[\text{Total errors}] \leqslant \varepsilon(T + (T-1) + (T-2) \ldots + 1) \propto T^2 . \tag{100}$$

To try to solve this problem, some researchers have proposed a human-in-the loop approach, where if possible, we should keep growing our dataset by including the actions that our expert would take along the paths taken by the computed policy by behaviour cloning.

Algorithm 11: DAGGER: Dataset Aggregation

```
Result: Best policy $\hat{\pi}_i$ on validation
Initialise $\mathcal{D} \leftarrow 0$
Initialise $\hat{\pi}_i$ to any policy
for i=1 to N do
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i$
    Sample T-step trajectories using $\pi_i$
    Get dataset $\mathcal{D}_i = (s, \pi^*(s))$ of visited states by $\pi_i$ and
      actions given by the expert $\pi^*$
    Aggregate datasets $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
    Train classifier $\hat{\pi}_{i+1}$
end
```

4.8.2 *Inverse RL*

The goal of inverse RL is to infer the reward function $R(s, a)$ given the demonstration trajectories. In this approach we have to make assumptions about the agent behaviour otherwise we cannot learn anything, in particular we need to assume that the policy followed by the teacher is optimal. However there is a problem, there are many reward functions that are consistent with the data.

4.8.2.1 *Linear Feature Reward Inverse RL*

In this approach we consider reward is linear over features and our goal is to find a weight vector $\mathbf{w}$ given a set of demonstrations.

$$R(s) = \mathbf{w}^T \mathbf{x}(s) . \tag{101}$$

DEFINITION 20: We represent as $\mu(\pi)(s)$ the discounted weighted frequency of state features under policy π defined as :

$$\mu(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \, \mathbf{x}(s_t) \,|\, \pi\right] , \tag{102}$$

and is something we can get purely relying on the demonstrations.

Recalling the original definition of the state-value function V^π and putting this together with Equation 101 and Equation 102 we obtain:

$$V^\pi = \mathbf{w}^T \mu(\pi) . \tag{103}$$

If we recall the partial ordering of policies we know that $V^* \geqslant V^\pi$ for every policy $\pi \neq \pi^*$. Assuming that our demonstrations are from the optimal policy, we want to find a weight vector $\mathbf{w}^{*T}$such that:

$$\mathbf{w}^{*T} \mu(\pi^*) \geqslant \mathbf{w}^{*T} \mu(\pi), \quad \forall \pi \neq \pi^* . \tag{104}$$

Algorithm 12: Apprenticeship Learning

```
Result: A policy that is as good as the expert policy $\pi^*$
Initialise policy $\pi_0$ to any policy
Initialise $i \leftarrow 0$
repeat
    $i \leftarrow i + 1$
    Find a reward function that the teacher maximally
      outperforms all previous controllers:
        $\arg\max_{\mathbf{w}} \max_{\gamma} \quad \text{s.t.} \quad \mathbf{w}^{*T}\mu(\pi^*) \geqslant \mathbf{w}^{*T}\mu(\pi) + \gamma$
        $\forall \pi \in \{\pi_0, \pi_1, \dots, \pi_{i-1}\}, \quad \|\mathbf{w}\|_2 \leqslant 1$        (105)
    Find optimal control policy $\pi_i$ for the current $\mathbf{w}$
until $\gamma \leqslant \varepsilon/2$
```

4.9 RECAP

During this section we explored very in depth many concepts of Reinforcement Learning. The objective of this introduction was to familiarise the reader, in a contextualised manner, with concepts like policy, the general RL formulation, TD error, the eligibility traces in SARSA(λ), actor critic architectures and the REINFORCE method, as several approaches for RL in SNN are inspired on such concepts. We also introduced Q-learning to later introduce Deep Q-Learning, and Policy Gradient algorithms to later introduce DDPG. These two approaches, DQN and DDPG, will be used in Chapter 7. We also decided to briefly talk about Imitation Learning which hasn't been explored with SNN and constitutes a good area for future work and approaches.

Part II

CONTRIBUTIONS

In this part we introduce the contributions obtained through the research done in this thesis. In particular, in Chapter 5, we clearly establish some links between RL, SNN and event-based cameras and briefly explore the domains of applications of this trio of bio-inspired approaches. In Chapter 6 we present feature extraction and object recognition, a field that could benefit from these approaches and propose a new approach that uses RL, SNN and event based cameras, while on Chapter 7, we present robot control and some contributions which also makes use of the trio of approaches.

5 BIO-INSPIRED TRINITY

In Chapter 2 we talked about spiking neural networks, while in Chapter 3 we talked about event-based cameras also called AER sensors, and in Chapter 4 we talked about Reinforcement Learning (RL). To answer the driving research question of this thesis, i. e. , is it possible to use at the same time the aforementioned trinity of bio-inspired approaches and technologies, and if so, where can they be applied, we have to analyse each one of the technologies from a biological perspective since it is the level at which everything was inspired and the commonalities or links between them are more likely to lie.

In the following sections we analyse the links between technologies and explicit the relationships between them, which is the first theoretical contribution of this thesis. In particular, in Section 5.1 we cover the link between event cameras to spiking neural networks , in Section 5.2 we link spiking neural networks to reinforcement learning, while in Section 5.3 we link event-based cameras to reinforcement learning. Finally, in Section 5.4 we take a look to all the three technologies/approaches together and briefly analyse how and in which fields they have been jointly used.

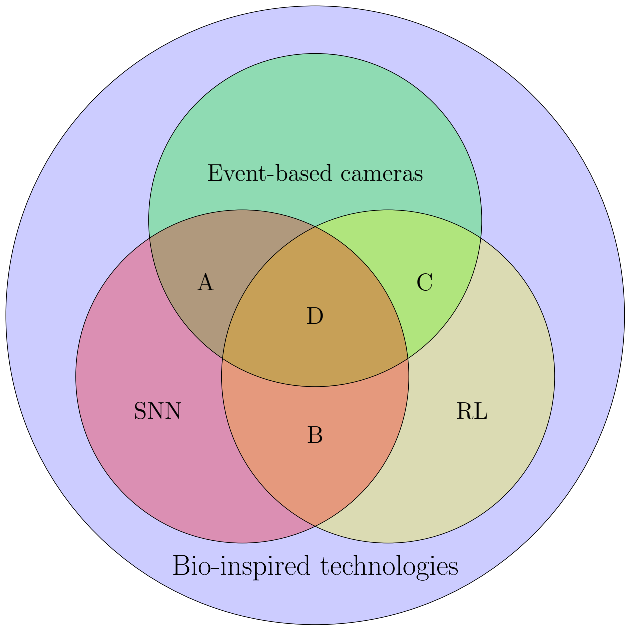


Figure 28: Triad of approaches and their intersections.
A) Section 5.1, B) Section 5.2, C) Section 5.3, D) Section 5.4.

As we will see, this trinity of approaches are perfectly matched to be used together, moreover, thanks to their energy efficiency provided by their bio-inspired nature, event-based cameras and SNNs are perfectly suited for applications like robot control where there is limited power supply, while RL ensures a simple and effective way of learning.

5.1 LINKING EVENT-BASED CAMERAS TO SNNS

Linking event-based cameras to SNNs is actually simple. Indeed all it takes is to consider event-based cameras as something that provides an input to spiking neural networks. From the biological point of view, event-cameras work just like the human eyes, they provide a visual input to the processing unit. In the biological case such processing unit is the brain, while in our case, the processing units are spiking neural networks.

Just like the brain is able to process also other signals coming from other senses (smell, touch, taste and hearing), spiking neural networks can be used to process additional information such as the one provided by the Inertial Movement Unit (IMU) in the DAVIS sensor.

Although the relationship between the two is very simple, the link we can draw between event-cameras and spiking neural networks is more significant than the one that can be drawn with a generic type of sensor and a generic type of processing unit. In the case of event-based cameras and spiking neural networks, both technologies operate in an asynchronous manner and mimic in a better way their biological counterparts. A mapping between biological structures and neuromorphic hardware is shown in Figure 29.

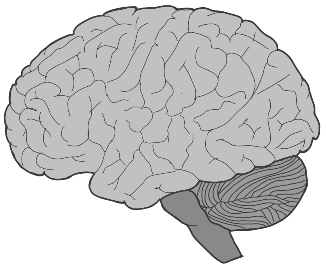
Event-based cameras and SNNs not only have the general link explored until now, in fact, they also have some very well-defined and bio-inspired links for particular applications. These special links were conceived to increase the biological realism in which both technologies operate with the final goal of architecting a complete neuromorphic system which replicates both the form and function of the brain.

Over the next subsections, we will take a look at two of the most common specialised links between event-based cameras and SNNs. In particular in Section 5.1.1 we explore the link for object recognition while on Section 5.1.2 we explore the link for motion estimation.

5.1.1 *The object recognition link:* HMAX

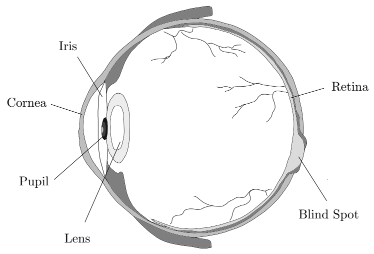
One of those well-constructed links between SNN and vision sensors like event-based cameras is the HMAX model for object recognition.

(a) A SpiNN-5 Development Board.



(b) A representation of the human brain.

(c) A Prophesee event-based camera.



(d) A representation of the human eye.

Figure 29: Left: neuromorphic hardware. Right: biological counterparts of neuromorphic hardware.

Initially proposed by Riesenhuber and Poggio [42] and later extended by Serre et al. [43], the HMAX model is a neurophysiologically plausible model of visual recognition which mimics the structure of the visual cortex. In its simplest form, this hierarchical feed-forward model consists of four layers of computation where so-called simple S units alternate with complex C units. These 4 layers imitate two of the structures of the cerebral cortex processing visual information: the Primary Visual Cortex (V1) corresponding to the first two layers of the model (S1 and C1), and the extra-striate Visual Area (V4) corresponding to the other two layers (S2 and C2).

The tasks accomplished by the layers are the following:

- The S1 layer consists of a battery of Gabor filters which have been shown to provide a good model of cortical simple cell receptive fields [44]. These filters are selected to respond to different orientations and scales.
- The C1 layers pool over position and scale of their S1 unit inputs through a maximum (`max`) operation, thereby increasing tolerance to shift and size.

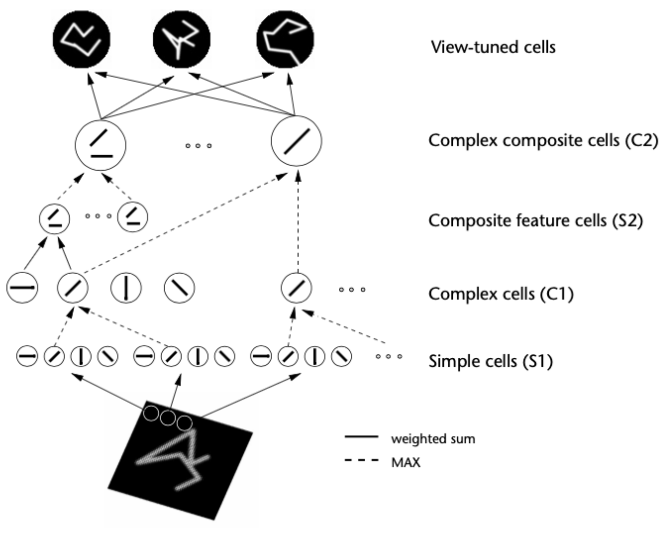


Figure 30: HMAX model. Image taken from Riesenhuber and Poggio [42].

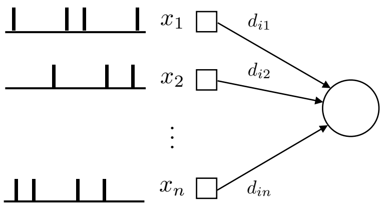
- The S2 layer combines responses from different orientations using weights that are obtained through some learning stage.
- The C2 layer is used to pool all responses from all S2 units for classification purposes, by taking a global maximum on the S2 responses.

There are more biologically complete models which include the infero-temporal cortex and the prefrontal cortex such as the extended model proposed by Serre et al. [43], however implementations usually use the four aforementioned layers.

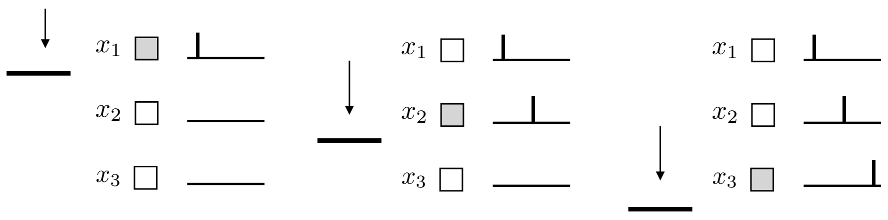
5.1.2 *The motion estimation link: Reichardt detectors*

Event-based vision sensors have some a special link to SNN for the motion estimation task. There are several approaches linking the two for this task: token methods, frequency methods and correlation methods, however, the most biologically plausible link is based on Reichardt detectors [45], which are hypothetical bio-inspired neural circuits for how the brain is able to track motion from a visual input.

As it can be observed in Figure 31, in such scenario, neurons have synaptic delays instead of synaptic weights and the delays are used to create coincidence detection of a particular motion. This connection allows the exploitation of the sparse high temporal resolution data provided by such sensors and the asynchronicity of the SNN model.



(a) Each neuron i has a different delay d_{ij} with each pixel x_j.

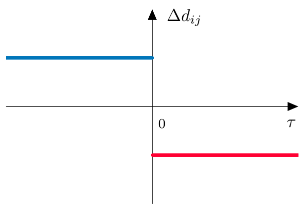


(b) Example of a movement of a bar from top to bottom at a certain speed. The movement causes the pixel to fire an event. By appropriately choosing the delays d_{ij}, we can make the signal from all pixels reach the neuron at the same time, e.g., if the bar moves at 1 pixel/ms the delays would be $d_{ij} = 3 - j$ milliseconds.

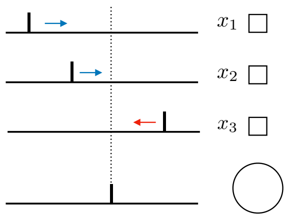
Figure 31: Synapse delays can be used for coincidence detection. An appropriate choice of the delays allows a neuron to be able to recognise motion at a given speed and direction.

One of the approaches based on this link that is worth mentioning, is the Spiking Architecture for Visual Motion Estimation (SAVME) [46]. This bio-inspired architecture consists on two layers made up of IF neurons with linear leakage where synapses are given a time delay to create a temporal pattern. It combines 64 neurons measuring movement at 8 speeds and in 8 different directions.

One of the disadvantages of the SAVME approach is that the delays are hard-coded in the architecture. To address this problem, such an architecture can be combined with a novel synaptic delay learning rules like Spike-Timing Dependent Delay Plasticity (STDDP) developed by Wang et al. [47]. This delay learning rule, which is heavily inspired by STDP, modifies synaptic delays instead of synaptic weights with the update window shown in Figure 32.



(a) The STDDP delay update function depending on the post-pre firing time delta τ.



(b) STDDP tries to adjust the delays so that the delayed input spikes become synchronous.

Figure 32: Spike-Timing Dependent Delay Plasticity (STDDP)

5.2 LINKING SNN TO RL

In this section, we now focus on linking reinforcement learning approaches to spiking neural networks. As it has been mentioned repeatedly during the previous chapters, both approaches are bio-inspired. For this reason, researchers have tried to link them by looking more in depth at how humans learn from the biological point of view. Most approaches linking both are based on modulation of neurotransmitters, which are the chemical messengers that transmit a message across a synapse.

In Section 5.2.1 we introduce three-factor rules, a model that generalises many RL approaches based on neuromodulation. In Section 5.2.2, we look more into detail modulation of a neurotransmitter called dopamine, and an approach which uses neuromodulation of dopamine together with the STDP learning rule. Finally, in Section 5.2.3 and Section 5.2.4, we briefly take a look at other ways RL is linked to SNNs.

5.2.1 *Three-factor Rules*

One of the most important learning rules for SNNs is Spike-Timing Dependent Plasticity (STDP), a two-factor Hebbian rule. Despite its success, this learning rule is not able to take into account the presence or absence of a reward signal which is one of the core aspects of RL theory and a key element to learning. In fact, animals learn by trial and error and by reinforcing a desired behaviour with a positive reward signal. It is thought that rewarding situations are represented by changes in the concentration of neuromodulators such as Dopamine (DA).

DEFINITION 21 (THREE-FACTOR RULES): Three-factor rules are a generalisation of Hebbian rules which consider the presence or absence of a modulatory signal [48]. These rules are called in some occasions *neoHebbian* rules. They have the following structure:

$$\Delta w_{ij} = \alpha(w_{ij}) \cdot M \cdot f(pre, post), \tag{106}$$

where pre denotes the spike train of a presynaptic neuron, $post$ the state of a postsynaptic neuron, M denotes a modulator and α a learning rate factor.

There are multiple ways of shaping the modulatory factor, we can use a reward signal R, a surprise/novelty factor S, the TD-error δ^{TD}, and so on. Generally, we distinguish three main theories of three-factor rules. The first one derives a three-factor learning rule by optimising through gradient descent and can be linked to policy gradient methods (Section 4.7). The second one uses a reward-modulated version of STDP (R-STDP), while the third one translates the framework

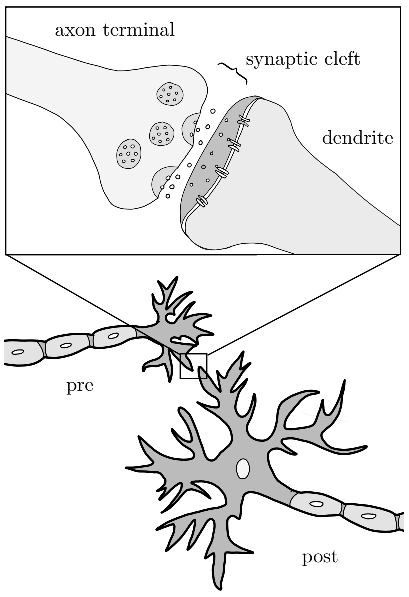


Figure 33: Release of neurotransmitters.

of Temporal Difference (TD) (Equation 53) and Actor Critic models (Section 4.7.2) to SNNs.

5.2.2 *Reward-modulated STDP*

Biological synapses release a type of chemical messengers called *neurotransmitters* upon the arrival of an action potential from the presynaptic neuron. As shown in Figure 33, these neurotransmitters are diffused across the small space between the two neurons which is called *synaptic cleft* and then bind to receptors on the dendrites of the post-synaptic neurons. As a result of this binding, the postsynaptic neuron is influenced in its behaviour and its spike-generating activity is either excited or inhibited.

It has been shown also that the release of a particular neurotransmitter called Dopamine (DA), is causally linked to the expected future reward [49]. Indeed, unexpected rewards activates midbrain dopamine neurons, and the magnitude of the dopamine release depends on unexpectedness of the reward.

The "credit assignment problem" is a central problem in the reinforcement learning literature. This problem consists in knowing how and which of the cues and actions received prior to the reward should be credited for it, even in situations where the reward may be delayed. Izhikevich [50] showed how modulating the Long Term Potentiation (LTP) and Long Term Depression (LTD) components of STDP with the neuromodulator DA is a reasonable solution to this problem.

The modulation of STDP with the neuromodulator DA proposed by Izhikevich [50] in 2007, transforms the standard unsupervised learning paradigm of STDP, into a reward-based learning paradigm, it is for this reason that is sometimes given the name of Reward-modulated Spike-Timing Dependent Plasticity (R-STDP). This modulation is made possible by keeping 2 variables to describe each synapse: the *synaptic weight/strength* s, and c, an enzyme important for plasticity that acts as an "synaptic eligibility trace", in addition to a global variable d that is used to describe the extracellular dopamine.

eligibility trace
$$\frac{\partial}{\partial t}c(t) = -\frac{c(t)}{\tau_c} + \text{STDP}(\tau)\,\delta(t - t_{pre/post}), \tag{107}$$

extracellular dopamine
$$\frac{\partial}{\partial t}d(t) = -\frac{d(t)}{\tau_d} + DA(t), \tag{108}$$

synaptic strength
$$\frac{\partial}{\partial t}s(t) = c(t)\,d(t). \tag{109}$$

As we can see from the equations above, the eligibility trace is a concept borrowed from the SARSA(λ) approach in classical RL theory (explored in Section 4.4). This trace decays with a time constant τ_c and suffers modifications according to the STDP [1] rule shown in Figure 10 and when the Dirac delta $\delta(t)$ is enabled at pre and post firing times $t_{pre/post}$, dopamine decays with a time constant τ_d and increases with a term DA that acts upon the arrival of the reward, and the synaptic strength is modified only when dopamine is present ($d > 0$), and when it is eligible for the reward ($c > 0$).

This approach has been employed by several researchers: Farries and Fairhall [51] employed this mechanism to train neurons to generate a particular pattern of spikes, while in 2009 Vasilaki et al. [52] explored its use on continuous space.

There are multiple implementations of R-STDP in different domains, however not all of them assign the neuromodulatory reward signal or use the dopamine modulation the same way. According to Meschede et al. [53], we can classify the approaches into:

① *Classical event-based reward learning*. This kind of learning is the one that resembles the most to the classical RL framework as it uses rewards associated to specific events.

② *Control error minimisation*. The dopamine-modulated learning is used to minimise an objective function, synapses are modified according to their eligibility trace and the impact on the objective function.

1 We recall that STDP(τ) indicates a function which returns a value depending on the post-pre firing delta also called inter-spike interval $\tau = t_{post} - t_{pre}$.

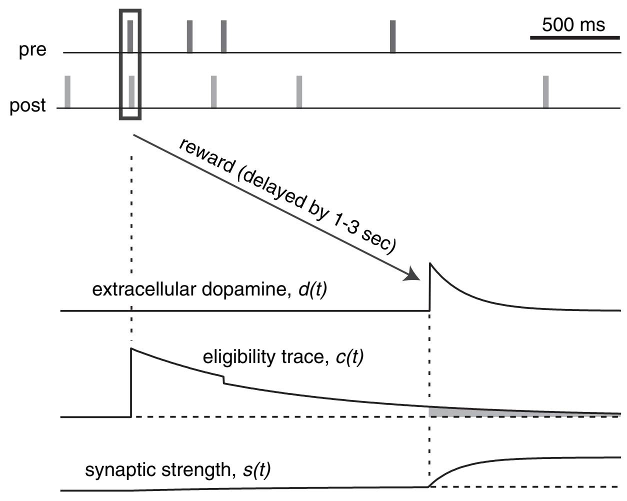

500 ms
pre
post
reward (delayed by 1-3 sec)
extracellular dopamine, *d(t)*
eligibility trace, *c(t)*
synaptic strength, *s(t)*

Figure 34: Dopamine Modulated STDP, taken from Izhikevich [50].

3. *Indirect control error minimisation*. Changes in the synapses are generated by a separate critic SNN.
4. *Metric Minimisation*. A global metric (which is easier to construct than a controller), is minimised.
5. *Reinforcing Associations*. As in classical conditioning, a dopamine-modulated synaptic plasticity rule was used to reinforce associations between conditioned and unconditioned stimuli.

5.2.3 *Stochastic synapses*

Operant conditioning is a process through which a behaviour is modified by means of a reward or a punishment. In such process, animals voluntarily increase the future probability of actions that were performed prior to a reward. One interpretation for this phenomenon is because animals are *hedonistic*, i. e. , reward seekers.

It also happens that chemical synaptic transmission is an unreliable process. As it has been shown by Stevens [54] in 1993, a presynaptic terminal may release a neurotransmitter, or may fail to release it.

Other theories linking RL and SNN such as the one proposed by Seung [55] in 2003, combine these two aspects: neuromodulation and unrealiability of synapses. In this model, synapses are unreliable and hedonistic (reward seekers), and learn by computing a stochastic approximation to the gradient of the reward with respect to the synaptic strengths.

Formally, in this model a synapse can be either available or refractory. When a presynaptic spike stimulates the release of neurotransmitters, it does so with a probability p:

$$p = \frac{1}{1 + e^{-q-z}}, \tag{110}$$

$$\frac{\partial}{\partial t} z(t) = -\frac{z(t)}{\tau_z} + \Delta_z \delta(t_{pre}), \tag{111}$$

where q is a release parameter, and z models the calcium dynamics at the presynaptic terminal. The parameter z decays with a time constant τ_z, and jumps by a constant Δ_z when the Dirac delta $\delta(t)$ activates at the arrival time of a presynaptic spike t_{pre}. In order to learn from reinforcement this type of synapse, this type of synapse must keep another variable with similar dynamics. Such variable c must acts as an eligibility trace and maintain a record of the synapse's recent releases and failures. The jump Δ_c is not constant but instead it is designed so that the eligibility trace has exactly zero mean:

$$\text{eligibility trace} \qquad \frac{\partial}{\partial t} c(t) = -\frac{c(t)}{\tau_c} + \Delta_c\, \delta(t_{pre}), \tag{112}$$

$$\text{eligibility jump} \qquad \Delta_c = \begin{cases} 1 - p, & release \\ -p, & failure \end{cases}. \tag{113}$$

If there is actually a release of neurotransmitters, the synapse enters a refractory state and recovers with time constant $1/\tau_r$.

Plasticity is driven by the product of a learning rate $\eta > 0$, a reward signal $h(t)$ and the eligibility trace $c(t)$:

$$\text{plasticity} \qquad \frac{d}{dt} q(t) = \eta\, h(t)\, c(t). \tag{114}$$

This approach is derived from the REINFORCE RL algorithm introduced in Section 4.7.1. Indeed the name is an acronym for the formula:

$$\begin{matrix}\text{REward} \\ \text{Increment}\end{matrix} = \begin{matrix}\text{Non-negative} \\ \text{Factor}\end{matrix} \times \begin{matrix}\text{Offset} \\ \text{Reinforcement}\end{matrix} \times \begin{matrix}\text{Characteristic} \\ \text{Eligibility}\end{matrix} \tag{115}$$

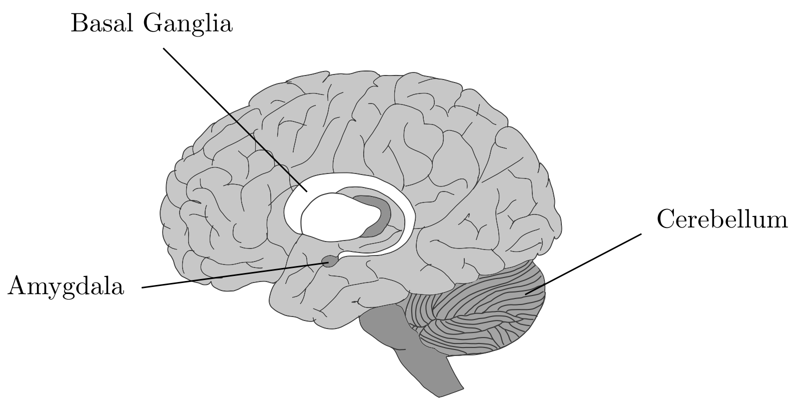


Figure 35: Basal Ganglia.

Note that the synapses obey to the following rules:

① The release probability is increased if rewards follows release and is decreased if reward follows failure.

② The release probability is decreased if punishment follows release and is increased if punishment follows failure.

It is thus the difference between the release-reward correlation and failure-reward correlation which indicates whether the release probability should be increased or decreased.

5.2.4 *Basal Ganglia*

Finally, the last way to link RL to SNN that will be explored is Basal Ganglia (BG). Basal Ganglia refers to a group of structures that constitute one of the brain's fundamental processing units. The word *basal* refers to the fact that the basal ganglia is found near the base, or bottom, of the brain. This group of structures is thought to be involved in a variety of cognitive, emotional, and movement-related functions, however, it is best-known for its role in movement.

In the Basal Ganglia, there is a region called Substantia Nigra pars compacta (SNc) that plays an important role in reward and movement. It contains dopaminergic neurons that release Dopamine neurotransmitters that can modulate the synaptic plasticity. This dopaminergic neurons thus allow a reward modulated synaptic modification.

Lately there have been some studies that link BG with the Actor-Critic architecture in RL. As shown by Schultz et al. [56], the activity of dopaminergic neurons seems to encode the difference between the expected reward and the actual reward in a way that resembles a lot the TD-error in RL (Equation 92). The authors point out that dopamine responses provide enough information to implement a simple policy.

For instance, we could take actions that are correlated with an increase in dopamine and avoid those with a correlated decrease in dopamine activity.

Vasilaki et al. [52] suggest the following synaptic update rule:

$$\frac{d}{dt} w_{ij}(t) = \alpha(w_{ij})\,(R(t) - b)\,\delta(t - t^f)\,e_{ij}(t)\ , \tag{116}$$

where δ is the Dirac delta activated at the spike firing time t^f, and $e_{ij}(t)$ is an eligibility trace that stores the correlation between pre- and post-synaptic activity. If we carefully analyse Equation 116, we can see that it has the same structure as the REINFORCE formulation (Equation 115) and that the term $(R(t) - b)$ resembles TD error.

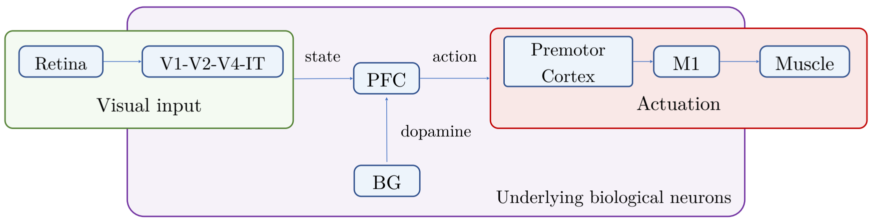
5.3 LINKING EVENT-BASED CAMERAS TO RL

As it was shown by Arakawa and Shiba [57], event-based cameras and reinforcement learning can be used together by means of a deep RL approach. In such approaches, an image-like feature can be created out of the stream of events to work as a state representation of the environment. In the experiments they carried out, they were able to demonstrate that a deep RL approach (in their case: double DQN) and an event-based camera, can in fact be used together. Moreover, it is possible to take advantage of the increased temporal resolution of the sensor to enable a much faster control loop which is not limited to the typical 30-60Hz rate at which conventional RGB-cameras operate, in addition to their power efficiency advantages.

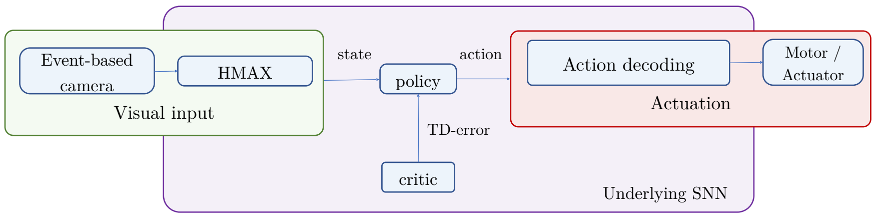
5.4 LINKING THE TRINITY

It is natural to think that a link between the three technologies is simply obtained by using data coming from an event-based camera on a spiking neural network which learns with a reinforcement learning mechanism, however there is a deeper link between the three technologies and is based on the biological way we learn to take actions. As it was explained in Chapter 4, reinforcement learning is not just about a notion of reward, but also about learning how to perform some actions in order to maximise a reward. If we look at RL from the big picture, we can establish some links that go beyond the learning mechanism but concentrate more at high level on the RL paradigm.

The human brain contains a structure called the Prefrontal Cortex (PFC), which is considered to be the brain region responsible for the orchestration of thoughts and actions in accordance with internal goals [58]. This region, resembles pretty much to a *policy* in the RL paradigm, since it is responsible of taking actions given a



(a) Biological control loop.



(b) Bio-inpired control loop.

Figure 36: Biological vs neuromorphic control loop. It is easy to see from the image above that a bio-inspired control loop can be achieved by mapping of real brain structures to bio-inspired components or concepts seen in previous sections or chapters.

state. The difference in this case, is that the aforementioned brain region works with a state representation which is given by some other brain structures after processing the information coming from the five senses, and the reward which is encoded through the release of neurotransmitters like Dopamine (DA) by the Substantia Nigra pars compacta (SNc) in the Basal Ganglia (BG).

Another difference between the PFC and a policy in a usual RL problem, is that the action stays "high level" until it is concretised by the Premotor cortex and the Primary Motion Cortex (M1). It is these brain structures which translate the "high level" actions into movements of the appropriate muscles.

Given the description of multiple brain structures in this chapter, we propose in Figure 36 a mapping of concepts which highlights the similarities between brain structures and models, and points how a possible bio-inspired control loop could be assembled with current models. A general overview of how actions are taken by humans is illustrated in Figure 37 by highlighting the relevant brain structures involved in such process.

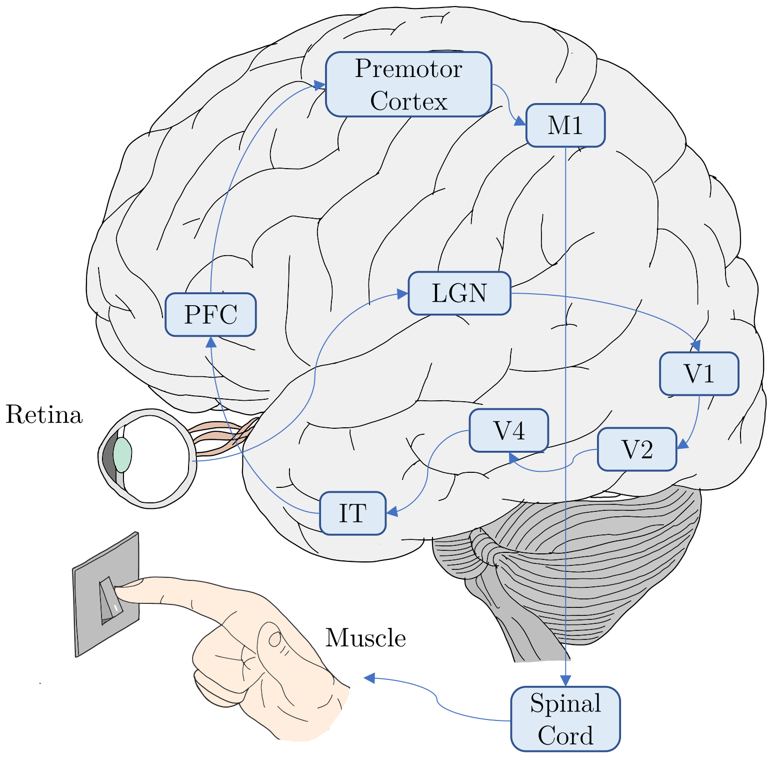


Figure 37: When a human has to take an action, the visual information flows to several brain structures, each of which perform a very particular action just to the point where an action is taken by the muscles of the body relevant to that action. The information flows from the retina to the Lateral Geniculate Nucleus (LGN) which is the main central connection for the optic nerve to the occipital lobe, particularly the Primary Visual Cortex (V1). The information then flows to other structures of the visual cortex: the Secondary Visual Cortex (V2), the Visual Area (V4) and the Inferior Temporal Cortex (IT) until the information finally arrives to the Prefrontal Cortex (PFC). Influenced by the learning suffered with neuromodulation of dopamine, the decision-making mechanism of the PFC selects an action to take and contretises it by means of the premotor cortex and Primary Motion Cortex (M1). Researchers estimate that this control loop is might be pretty slow and takes about 250ms to complete [59].

It is evident that the trio of technologies can be applied in situations where robot control is involved, in addition, as bizarre as it may sound, this trio of technologies can be used also for feature extraction and object recognition. This is done with inspiration of the underlaying working principle of the IT, which is involved in processing visual stimuli and identification and recall of objects and where "actions" are not concretised but simply represent a decision.

In Chapter 6, we analyse feature extraction and object recognition and propose, to the extent of our knowledge, the first approach combining these three technologies, while in Chapter 7, we take a look at applications in robot control and benchmark a novel RL approach for an UAV relying on SNN having as input data event-based cameras, with more traditional approaches. In these chapters we will combine the technologies not with a one-to-one mapping of concepts as shown in Figure 36 but with approaches better suited for each application.

6 FEATURE EXTRACTION AND OBJECT RECOGNITION

The novel technologies introduced in Part I have potentially many diverse applications. One of the research fields out of the many that can really benefit from Event-based Cameras, Spiking Neural Networks, and Reinforcement Learning, is Feature Extraction and Object Recognition. The chapter is structured as follows. In Section 6.1, we introduce some of the already existing approaches and the current state of the art in Feature Extraction and Object Recognition for event-based data. Then in Section 6.2, we build a new approach which takes ideas from previous approaches, but appropriately combines them with a RL-inspired learning rule to form a novel approach for object recognition combining the three bio-inspired technologies. Appendix E contains a paper submitted for review at CVPR [1] summarising the novel approach introduced in this chapter.

6.1 CONTEXT AND STATE OF THE ART

Feature extraction is a field of application that builds derived values from the input data, these derived values are called *features* and intend to be a set of informative and non-redundant values, aiming to facilitate subsequent learning and generalisation steps. A common task is the extraction of visual features in images for object recognition in the visual scene. In such scenario, a set of features is extracted in a way that their activations/presence facilitates recognition of the particular object represented on the image in subsequent classification and detection algorithms.

There are many feature extraction and object recognition techniques for standard images, however, as it has been explained, event-based cameras introduce a new paradigm and previous approaches cannot be used. In the following sections, we take a look at some of the approaches for event-based data that have been developed since the spread of the technology in the scientific community in late 2000's/early 2010's, most of them relying on the HMAX model (cfr. Section 5.1.1).

Approaches in this field can be thought to perform the same function of the Inferior Temporal Cortex (IT), which is a key region of the ventral visual pathway in the human brain implicated in visual processing and visual object recognition.

1 The Conference on Computer Vision and Pattern Recognition (CVPR) is an annual conference on computer vision and pattern recognition, which is regarded as one of the most important conferences in its field.

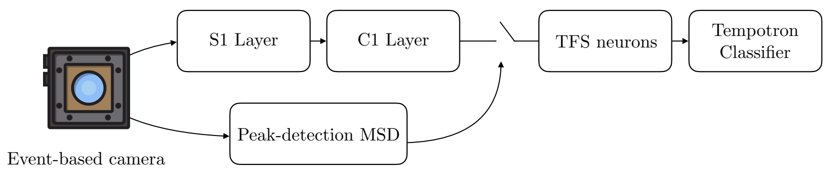


Figure 38: EDHMAX Architecture.

6.1.1 EDHMAX

One of the first approaches for object classification using AER data was introduced in 2014 by Zhao et al. [39]. This approach achieves object classification by extracting cortex-like features with a simple two-layered HMAX-inspired model [42] followed by a Tempotron classifer [60]. Instead of processing data with a fixed number of events, this approach is event-driven, which means that there is no clock and events are processed continuously. Due to the aforementioned characteristics we have decided to introduce EDHMAX (Event-Driven HMAX) as an unofficial acronym for this work, which makes further references to this work a lot easier.

The two feature extraction layers in the model do not correspond to the a complete model of the visual cortex (V1-V2-V4-IT) as the extended HMAX model, but instead, this approach uses the simplified architecture in the following way:

- an S1 convolutional layer that performs pattern matching on 4 scales and 4 orientations. Pattern matching is applied with Gabor filters and are equipped with a constant linear leaking forgetting mechanism that allows continuous processing.
- a C1 layer that encodes competition among neurons in a receptive field by applying a `max` operation.

A peak-detection Motion Symbol Detector (MSD) is introduced to perform a snapshot of the C1 features, these features are encoded using an I2L encoding, where the higher the response, the shorter the time to first spike (cfr. Section 2.3.2).

As a classifier, this approach uses the Tempotron classifier that implements a fire/not-fire learning in which updates to the weights are done according to whether the class neuron should have fired or not, i. e. if the neuron fires correctly (incorrectly), the weights of the afferent synapses are increased (decreased). In order to reduce complexity, this approach uses a Look-Up-Table (LUT) to store the weights and use or update the few of them that are relevant, only when they are needed. An overview of the approach can be seen in Figure 38.

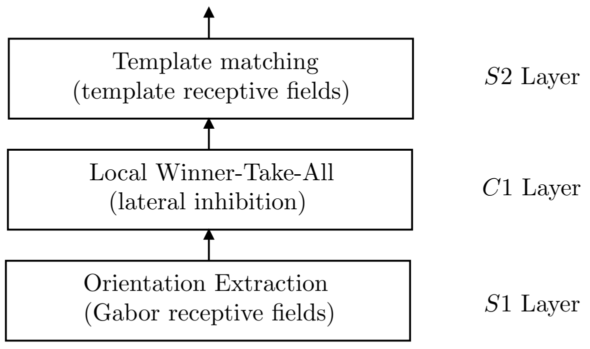


Figure 39: HFirst Architecture.

6.1.2 HFIRST

HFirst is a spiking hierarchical model for object recognition developed by Orchard et al. [36]. It was given such a name because instead of relying on a non-linear pooling operation such as the `max` operation in standard frame-based CNNs, the approach relies on the first spike received during computation. Whilst the `max` operation outputs a number representing the strength of the strongest input, the first spike not only takes this into account (since strongly activated neurons tend to fire first), but also preserves the time encoding of signal strength. This simple temporal winner-take-all (WTA) mechanism is used to derive temporal features for object recognition that are both robust and have low computational cost.

HFirst is also heavily inspired from the popular HMAX hierarchical neural model [42], and consists of four layers: Simple 1 ($S1$), Complex 1 ($C1$), Simple 2 ($S2$), and Complex 2 ($C2$). The $S1$ layer performs orientation extraction with Gabor filters at 12 orientations (every 15 degrees) and is followed by the $C1$ layer which performs the first-spike pooling operation. The $S2$ layer combines responses from different orientations whilst $C2$ layer pools across all $S2$ spatial locations. Usually, the $C2$ layer is not used when there are multiple objects of interest in the scene, as the layer would discard information regarding the location of the object. An overview of the approach is in Figure 39.

The model also implements lateral inhibition to block responses from other neurons in the same pooling area.

The training consists in using hardly-segmented events (with jAER[2] [38]) and counting the number of spikes of each C1 neuron for the input sample. The count is translated to synaptic weights for the S2 layer recognising a particular class.

2 jAER Github project: https://github.com/SensorsINI/jaer/

6.1.3 BOE

Bag Of Events (BOE) is a feature extraction method for data retrieved with event-based cameras developed by Peng et al. [37]. Instead of lines, corners, or other visual features, this method uses the joint probability distribution of the consecutive events to represent the input. Moreover, the method uses a soft event segmentation instead of a hard event segmentation, meaning that it will not group the events using a fixed time slice or fixed number of events, instead it uses a more flexible approach through a LIF neuron. One of the advantages of this algorithm is the fact that it is an online learning algorithm, thus it does not require the entirety of the training data set to be provided in advance.

The learning process has two fundamental components, one of those is a Motion Symbol Detector (MSD), which is the LIF neuron used to regulate the learning process. At each input event, the Post Synaptic Potential (PSP) of the neuron increases, and once the neuron fires (i. e., when the PSP has exceeded a threshold), the learning process is triggered. To be more precise, after the neuron fires, all the events received since the last spike are grouped in the second fundamental component: the Segment Recorder (SR). This component keeps count of the number of times each pixel emitted an event in a vector called segment (thus it is a $1 \times m$ vector where m is the total number of pixels in the sensor). Segments can be seen as a bag of events (hence the name) and can be represented as the joint probability distribution of the events $e_1 \dots e_k$ in a segment s_j:

$$s_j = P(e_1, e_2, \dots, e_k) \quad \forall e_i \in s_j \ . \tag{117}$$

By assuming the occurrences of the events in segments are statistically independent, authors propose an event frequency representation $f_{1j}, f_{2j}, ..., f_{mj}$ as a measure for popularity, where f_{ij} is the frequency of event e_i within segment s_j. This representation alone is insufficient for classification purposes, for this reason they propose a method that has the advantages of the popularity and speciality measures:

$$q_{ij} = w_i \, f_{ij} \quad \text{where} \quad w_i = -\log \frac{n}{n_i} \ , \tag{118}$$

where w_i is the self-information measuring speciality, f_{ij} is the frequency representation introduced before which measures popularity, n_i is the number of segments where an event e_i appears, and n is the total number of segments. Note that this method is evidently inspired from the `tf-idf` weighting scheme in information retrieval [61]. If an event e_i appears in all segments, its self-information is zero.

This feature extraction method can be paired with a classifier to perform object recognition. Authors used a linear Support Vector Classifier.

6.1.4 HOTS

HOTS is a hierarchical time-oriented approach developed by Lagorce et al. [62] that extracts spatio-temporal features called *time-surfaces* from the asynchronously acquired dynamics of a visual scene. These time surfaces are created with the process shown in Figure 40.

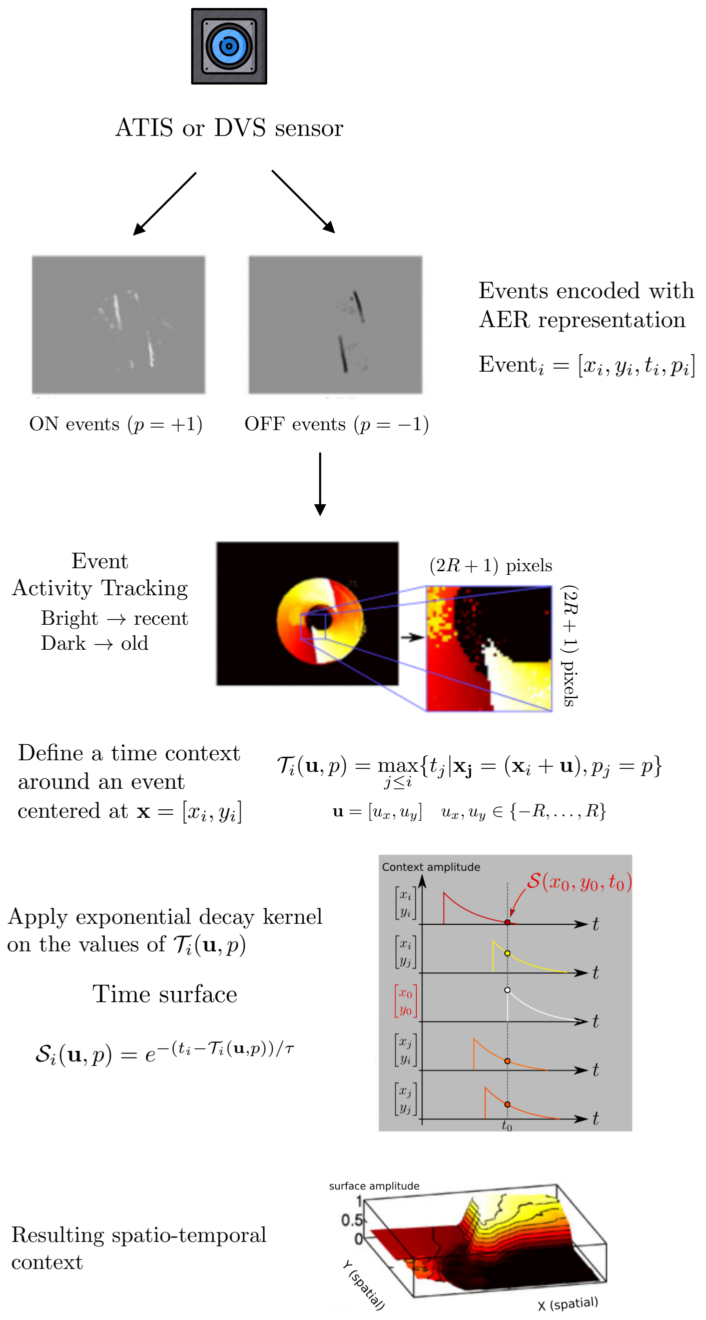

ATIS or DVS sensor
Events encoded with AER representation
$\text{Event}_i = [x_i, y_i, t_i, p_i]$
ON events ($p = +1$)
OFF events ($p = -1$)
Event Activity Tracking
Bright → recent
Dark → old
$(2R+1)$ pixels
$(2R+1)$ pixels
Define a time context around an event centered at $\mathbf{x} = [x_i, y_i]$
$\mathcal{T}_i(\mathbf{u}, p) = \max_{j \le i}\{t_j | \mathbf{x_j} = (\mathbf{x}_i + \mathbf{u}), p_j = p\}$
$\mathbf{u} = [u_x, u_y] \quad u_x, u_y \in \{-R, \dots, R\}$
Apply exponential decay kernel on the values of $\mathcal{T}_i(\mathbf{u}, p)$
Time surface
$\mathcal{S}_i(\mathbf{u}, p) = e^{-(t_i - \mathcal{T}_i(\mathbf{u},p))/\tau}$
Context amplitude
$\mathcal{S}(x_0, y_0, t_0)$
t
t_0
Resulting spatio-temporal context
surface amplitude
1
0.5
0
Y (spatial)
X (spatial)

Figure 40: The process of creating a time-surface in an R neighbourhood. Process shown only for one polarity, however a time surface is composed of two halves corresponding to both polarities of events.

Out of each incoming event, we construct a different time-surface. We can think of learning a set of elementary time-surfaces from the scene which can roughly represent the observed time-surfaces; such learned time-surfaces go under the name of prototypes. For the creation of these prototypes we can use a clustering algorithm such as the one proposed by the same authors, which classifies an event with the closest time-surface prototypes at that moment.

Algorithm 13: Online Clustering of Time-Surfaces

```
Initialize $p_n \leftarrow 1$, use first N events as initial values for $\mathbf{C}_k$
foreach incoming event $ev_i$ do
    Compute time-surface $\mathcal{S}_i$
    Find closet cluster center $\mathbf{C}_k$ (euclidean distance)
    $\alpha \leftarrow 0.01/(1 + p_k/20000)$
    $\beta \leftarrow \mathbf{C}_k \cdot \mathcal{S}_i/(\|\mathbf{C}_k\| \cdot \|\mathcal{S}_i\|)$
    $\mathbf{C}_k \leftarrow \mathbf{C}_k + \alpha(\mathcal{S} - \beta\mathbf{C}_k)$
    $p_k \leftarrow p_k + 1$
end
```

Once the prototypes have been learnt, the stream of input events can be transformed into a stream of prototype activations $\mathtt{feat}_i = [x_i, y_i, t_i, k_i]$ where k_i is the index of the prototype.

The paper also shows that time-surfaces can be arranged to form a hierarchical structure, where each layer computes a set of time surfaces, with the only difference being that hidden-intermediate layers take as input the temporal activity of the previous layers and have different constants for space-time integration of features defined by the following formulas:

$$\begin{aligned} R_{l+1} &= K_R \cdot R_l \ , \\ \tau_{l+1} &= K_\tau \cdot \tau_l \ , \\ N_{l+1} &= K_N \cdot N_l \ , \end{aligned} \tag{119}$$

where R_l defines the size of the time-surface neighbourhood, τ_l defines the time constant for the exponential kernel, N_l defines the number of prototypes learnt per layer and K_R, K_τ, K_N are parameters for the hierarchy.

We can build a histogram $\mathcal{H}$ that counts how many times a feature has been activated between instants t_{start} and t_{end} when an object arrives, this will constitute their *signature*. This hierarchical approach that learns features *is unsupervised* and paired to a classifier gives very good results on the standard POKER-DVS [63] and DVS-BARREL [36] datasets, however it is quite slow as the last layer — which is the input to the classifier — integrates information over the longest period of time in the hierarchy ($\simeq$ 1s).

6.1.5 MuST

The Multiscale Spatio-Temporal feature representation (MuST), just like BOE, also uses the concept of Motion Symbol Detector (MSD) introduced by Zhao et al. [39] which uses a LIF neuron to achieve a soft event segmentation, using peak detection in a certain time window. Despite this similarity, the proposed approach streams the events segment by segment towards a couple of layers of the bio-inspired HMAX model architecture developed by Riesenhuber and Poggio [42], which mimics the simple and complex cells in primary visual cortex, an S1 layer (4 different Gabor filters on 4 scales) and then a C1 layer (max pooling).

The C1 features are then forwarded to a set of encoding neurons that code features into the form of spikes with log I2L (cfr. Section 2.3.2) and merge the multiple scale responses. This approach then detects features in an unsupervised manner with STDP. After training, each neuron is assigned to a class based on its sensitivity to patterns of different classes. The final prediction for this algorithm is determined by averaging the firing rates of trained neurons per class and choosing the class with the highest average firing rate.

6.1.6 *Towards a new method*

In Section 6.1, we briefly saw some methods for feature extraction and object recognition on event-based data and we briefly introduced some approaches. The question we seek to answer is whether RL can be applied or not for such a task. To do so, we briefly analyse some SNN-only approaches which do not use event-based camera data, but may be useful to create a new approach.

One of the first approaches to make use of SNNs for feature extraction and object recognition with SNNs was developed by Masquelier and Thorpe [64] in 2007. This approach took inspiration from the widely imitated HMAX model [42] and applied visual feature extraction to natural (frame-based) images. In order to learn the feature extraction, the approach used the unsupervised learning algorithm STDP to learn the synaptic weights between the C1 layer and the S2 layer. Combined with a temporal coding scheme where the most strongly activated neurons fire first, this approach leads to a situation where neurons in higher layers become selective to frequently occurring feature combinations and thus work as feature extractors after training.

This approach clearly inspired MuST (Section 6.1.5), a modified version that adapted the method for event-based camera sensors and took a multiscale approach for feature extraction.

The surge of RL approaches inspired Mozafari et al. [65] to build a feedforward convolutional SNN which was able to perform object recognition on natural images with an RL-inspired rule. This learning rule, called Reward-modulated Spike-Timing Dependent Plasticity (R-STDP) is shown in Equations 120, 121. In simple words what this learning rule does is to invert the polarity of the STDP learning rule when a punishment is received.

$$\text{if } \mathit{reward} \qquad \Delta w_{ij} = \begin{cases} A_r^- \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) \geqslant t_i \\ A_r^+ \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) < t_i \end{cases} \tag{120}$$

$$\text{if } \mathit{punishment} \qquad \Delta w_{ij} = \begin{cases} A_p^- \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) \leqslant t_i \\ A_p^+ \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) > t_i \end{cases} \tag{121}$$

Unfortunately, the name assigned to this learning rule clashes with the dopamine-modulated learning rule, leading to a great confusion in the literature of which learning rule we refer to with R-STDP. To momentarily solve the clashing problem, in the context of object recognition we will refer to this learning rule when talking about R-STDP, while in the robot control domain which will be explored in the next chapter, R-STDP will refer to the dopamine-modulated learning rule (crf. Section 5.2.2).

This is not a very common application for RL, but while STDP performs well in the detection of statistically frequent features, it fails at detecting rare features, which might be important for decision making. Instead of extracting features and then performing categorization with an external classifier, this approach uses class-specific neurons that are reinforced to fire as early as possible if their target stimulus is presented.

Some interesting aspects emerging from experiments on this network is the fact that R-STDP increases computational efficiency with respect to STDP and it enables the network to find spatial and temporal features that are very informative for the decision making process. Moreover, with this learning rule, it is possible to readjust the behaviour of neurons should the target change, even after learning to recognise a very different behaviour (i. e. , it is possible to unlearn and learn the new rewarding behaviours online).

The approach uses a four-layered network based on the model developed by Masquelier and Thorpe [64] with two simple and two complex layers. In fact, both models are very similar, but such similarities have a reason: Timothée Masquelier, a researcher at CNRS Toulouse, is a co-author of both papers.

- *Layer 1* ($S1$): is a simple layer that converts the input image into spike latencies by detecting oriented edges with 4 Gabor filters and applying an I2L encoding scheme (i. e. the more salient an edge, the earlier its corresponding spike is propagated).
- *Layer 2* ($C1$): is a complex layer that performs local pooling on $S1$ introducing some position invariance and reducing the number of required neurons. It also implements two kinds of lateral inhibition: within a same orientation and between orientations.
- *Layer 3* ($S2$): is a simple layer with IF neurons that detects complex features. Neurons in this layer apply the R-STDP learning rule on the input spikes and apply a WTA algorithm in which the neuron that fires first is seen as the winner and is the only one to suffer changes on its synaptic weights.
- *Layer 4* ($C2$): is a complex layer that performs the decision-making. Each neuron is assigned to a category and performs global pooling on $S2$ based on the first neuron firing. The reinforcement (punishment) is performed if the decision is correct (incorrect).

One of the main drawbacks of this approach is that it is still for standard grayscale images. It is thus interesting is to explore if it is possible to adapt the R-STDP approach to work for event-based data, in a similar way as MuST (Section 6.1.5) did it with the STDP approach and extend it for multi-class classification. At the moment is of great relevance and innovativeness as the MuST approach was published very recently (March 2020), close to the time when this work started.

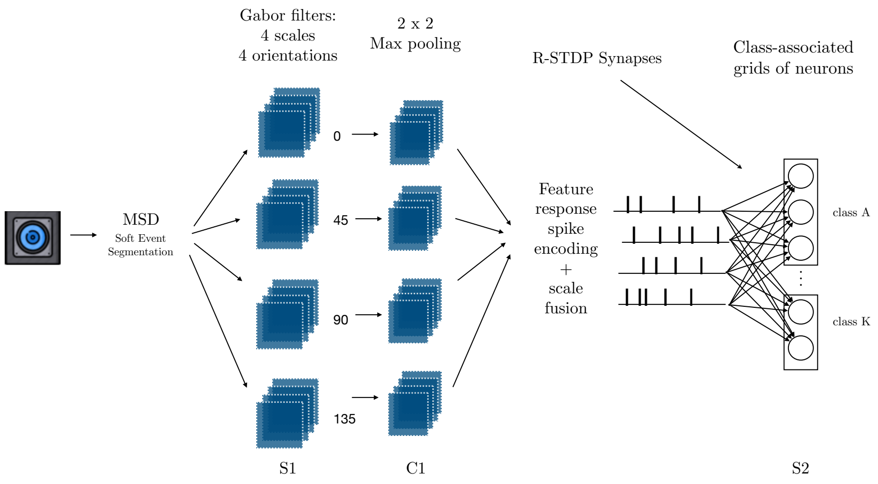


Figure 41: Overview of the ReMuS approach.

6.2 OUR NEW APPROACH: REMUS

The Reward-modulated MultiScale (ReMuS) approach, is our novel proposed feature extraction and object recognition approach that takes ideas from previous methods and combines them together. As it can be observed from the Figure 41, the approach consists of several phases and different components. The overall structure is inspired in the frequently used HMAX model structure and used similarly as in the HFirst approach (Section 6.1.2). The event segmentation idea is taken from BOE (Section 6.1.3), the multiscale approach is taken from MuST (Section 6.1.5) while the RL-inspired learning algorithm was heavily inspired on the algorithm developed by Mozafari et al. [65] (Equation 120) but is modified to prevent a dead-neuron problem.

Before delving deeper in the inner mechanisms of the approach, we briefly list the different phases of the method:

① The first step of the approach consists in grouping the events using a SES approach; once grouped, we generate a grayscale frame by integrating the events (Section 6.2.1).

② The grayscale images are convolved with 16 different gaussian filters, each of which detects a particular orientation at a certain scale. Max-pooling is then performed on the feature responses on a 2x2 region (Section 6.2.2).

③ The feature responses are encoded into spikes and scale-fusion is performed by combining information of spikes of different scales (Section 6.2.3).

④ The spikes are used as input for a SNN which has neurons covering a certain receptive field and implement a special kind of synapses whose weights are trained in an RL-inspired manner.

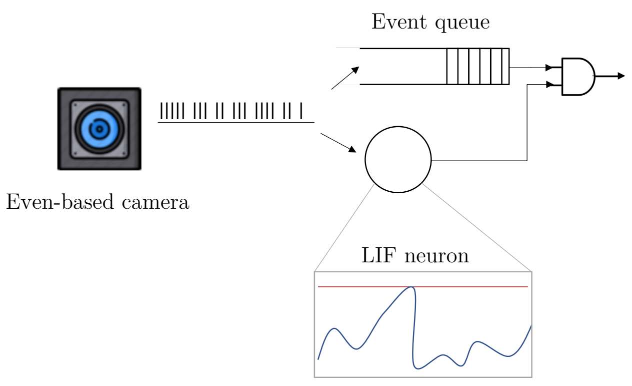


Figure 42: A threshold-based Motion Symbol Detector (MSD) used to implement a SES approach.

> The first spike coming out the SNN for each pseudo-frame is a partial prediction for the sample. The final prediction for the sample is the most predicted class out of all the predictions for the pseudo-frames in the sample (Section 6.2.4).

Now that we have a global overview of the approach, it is time to look more into details each of the phases.

6.2.1 *Pseudo-frame creation*

The first part of the processing pipeline consists in grouping the events in some way in order to be able to recreate a pseudo-frame from event. In this approach, a SES approach is employed to dynamically segment events into groups, in particular, a threshold-based Motion Symbol Detector (MSD) [39] is used.

The MSD illustrated in Figure 42 is composed of two components, an event queue and a LIF neuron. When an event is produced, the event will be routed to two sub components, an event queue which will accumulate events and the second component, the LIF neuron that will tell the queue when it is time to wrap up all the accumulated events in a batch and start all over again. At the arrival of an event, the neuron modifies its internal state by incrementing its potential by a certain quantity. This neuron is however susceptible to leakage, that means the potential suffers an exponential decay with time. This process is repeated for all incoming events. Whenever the threshold is reached, all the events accumulated in the queue are grouped together in a batch and sent to next step in the pipeline, the queue is cleared, and the neuron potential reset.

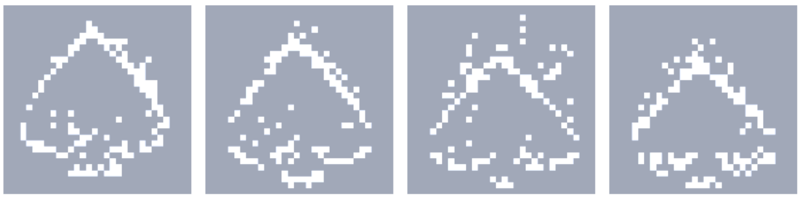


Figure 43: Reconstructed pseudo-frames with a MSD ($\tau = 0.02, \theta = 150$) of a POKER-DVS sample.

Once the events are grouped, we create a pseudo-frame by integrating the events. This pseudo-frame ignores the polarity and the temporal information of the events, the information left are the pixel coordinates of the event. Examples of this pseudo-images are shown in Figure 43.

6.2.2 *Pseudo-frame processing*

The pseudo-frames are processed with 16 different Gabor filters located in layer S1 to compute strength of spatial features. The 16 filters correspond to filters responding to 4 different orientations and 4 different scales.

The Gabor filters were obtained with the following formulas:

$$g(x, y; \lambda, \theta, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda}\right) , \tag{122}$$

$$x' = x \cos\theta + y \sin\theta , \tag{123}$$

$$y' = -x \sin\theta + y \cos\theta , \tag{124}$$

where λ represents the wavelength of the sinusoidal factor, θ represents the orientation to which the filter is sensitive to, σ represents the standard deviation of the Gaussian envelope and finally γ is the spatial aspect ratio which specifies how elliptical is the support of the Gabor function.

The parameters aspect ratio γ, effective width σ and wavelength γ assume commonly used values in the community which are the result of tuning efforts in previous work [43, 66]. The parameters are listed in Table 2.

filter size	3	5	7	9
effective width σ	1.2	2.0	2.8	3.6
wavelength λ	1.5	2.5	3.5	4.6
aspect ratio γ	0.3			
orientations θ	0°, 45°, 90°, 135°			

Table 2: Parameters used on Gabor filters.

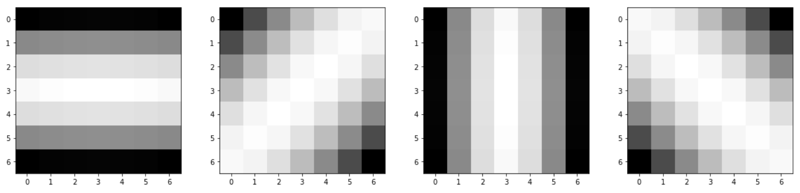


Figure 44: Gabor filters of scale 7 and orientations $\{0, 45, 90, 135\}$ degrees respectively.

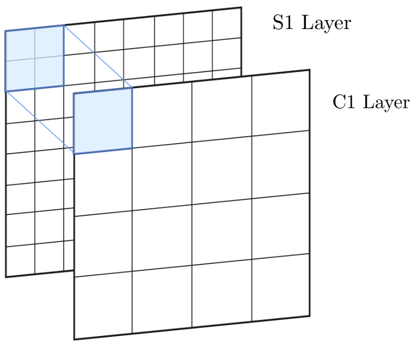


Figure 45: The C1 layer is responsible of pooling the S1 feature responses.

After convolving the pseudo-frame image with the 16 filters of which an example can be seen in Figure 44 with a `same` padding (meaning that the output image has the same size of the input image), the C1 layer performs max pooling over non-overlapping 2x2 regions of the feature responses as shown in Figure 45.

The pooling is done on each of the 16 resulting gaborised images showing the response for a particular scale and orientation. This process reduces both variance and dimensionality of the convolved image, being this last one effectively reduced half the original height and width.

6.2.3 *Spike encoding and multiscale fusion*

In order to transform the pooled feature responses into spikes, we need an encoding mechanism. In this case, we have several possible encoding mechanisms (cfr. Section 2.3.2). In this case, we choose logarithmic I2L, which is able to spread high and low response values better than linear I2L. In fact, it has been shown that this encoding mechanism has a higher information entropy H_{log} than linear I2L (meaning it is more informative) [20]. A general overview of the encoding approach can be observed in Figure 46, while a plot of I2L and logarithmic I2L functions can be observed in Figure 47

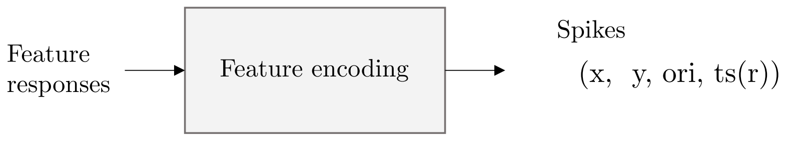


Figure 46: Feature encoding. The spike timestamp ts depends on the feature response value r. (cfr. Section 2.3.2)

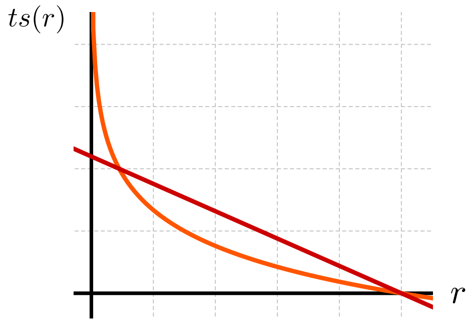


Figure 47: In red I2L (Equation 14), in orange logarithmic I2L (Equation 15). The values where both curves intersect corresponds to a value in the x axis which corresponds to minimum response r_{min} and on the y axis corresponds to the time window t_w. They also intersect on $(r_{max}, 0)$.

Similarly to MuST (Section 6.1.5), once the responses have been converted to spikes, we do not keep track of the scale at which the response originated, thereby performing a multi-scale fusion.

6.2.4 SNN

Once the feature responses have been encoded into spikes, the spikes are processed with an SNN. This SNN has IF S2 neurons, which cover a certain receptive field. The ensemble of neurons covering all the input area is called a *grid*. All neurons in the grid share the same weights in order to be able to detect that pattern in the whole input region. In order to achieve a distinction between the different classes, we need multiple grids of neurons, k per class with $k \geqslant 1$.

The S2 neurons in a grid increase their potential whenever there is a spike in their receptive field. The increase in the potential is given by the synaptic weight of the grid in that relative position.

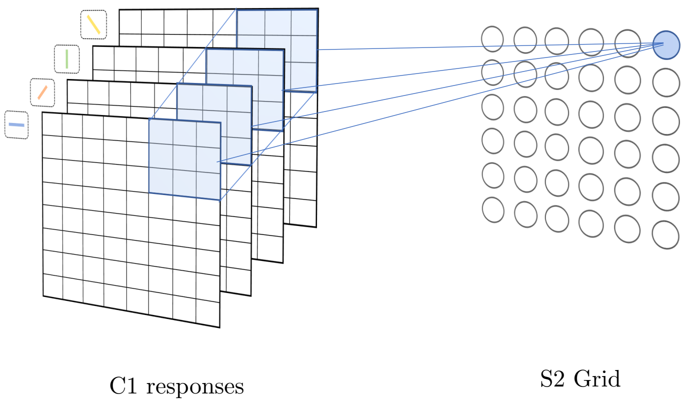


Figure 48: The S2 grid contains neurons which cover the entire C1 response region. In the particular case shown in the figure above, the grid weights are represented as a 3x3x4 matrix which covers a 3x3 region of 4 different orientations.

The first neuron in any grid that reaches the threshold and spikes is considered the winning neuron in a WTA setting. Since each grid is associated to a class, the prediction is the class where the spiking neuron belongs or silence in case of no spike. Formally:

$$\begin{aligned} \hat{y} &= \mathrm{label}(\mathrm{Grid}_m) \Leftrightarrow \exists j \forall l,\ t^f(j) \leqslant t^f(l) \wedge j \in \mathrm{Grid}_m \\ &\bigvee \\ \hat{y} &= \emptyset \Leftrightarrow \nexists j, t^f(j)\,, \end{aligned} \tag{125}$$

where m is the index of a grid $\in \{0, K-1\}$, j and l indicate S2 neuron indices and $t^f(\cdot)$ indicates the firing time of a neuron.

It is important to remark the fact that the only difference in each grid of neurons is the ensemble of weights which regulates the sensitivity to the C1 responses.

It is possible to get inspired from the RL-inspired learning rule developed for static images by Mozafari et al. [65]. However, in this approach, wrong predictions can lead to a heavy decrease in the weights which can ultimately result in grids with not spikes at all. Since Mozafari's rule accepts silence as a valid prediction, and does not update weights whenever there is a silence, this learning rule is subject to a "dead neuron" problem. We believe this is the reason why the original paper only tackled two-class classification problems. In a multi-class classification scenario, the method ultimately reached a point in which it was silent for most classes.

To address this problems, we propose a variation of the existing learning rule (Equation 120), which does not only update the grid of the neuron that spiked, but it also may increase all the weights of the grids associated to the correct class.

In case of a reward (correct prediction), the grid weights update is given by the following equation:

$$\Delta w_{ij} = \begin{cases} A_r^- \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) \geqslant t_i \\ A_r^+ \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) < t_i \end{cases}, \tag{126}$$

while in case of punishment (wrong prediction), the weight updates are instead the following:

$$\Delta w_{ij} = \begin{cases} A_p^- \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) \leqslant t_i \\ A_p^+ \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) > t_i \end{cases}. \tag{127}$$

In addition, if there is a wrong prediction or silence,

$$w_{ij} + \epsilon, \qquad \forall w_{ij} \in \mathtt{Grids}(\mathtt{label}) \,. \tag{128}$$

With these update rules, if a reward is received (Equation 126), only the grid containing the neuron that spiked undergoes a weight modification. This update rule increases the weights by a factor of $A_r^+ > 0$ if at a certain position in the neuron's receptive field, there was a C1 response spike before the S2 neuron spiked (successful situation: true positive). Given that the prediction was correct (since a reward is received), this increase in the weights aims to reinforce the association of certain features' position and orientation to the class. In parallel, this same update rule updates weights by a factor of $A_r^- < 0$, if at a certain position, there was not a C1 response spike before the S2 neuron spiked (silence: true negative). This decrease in the weights is triggered so that S2 becomes sensitive only to relevant positions and orientations, and ignores the rest of the spikes given that they might not be class-relevant.

When instead a punishment is received (Equation 127), the weights are updated by a factor of $A_p^- < 0$ if at a certain position, there was a C1 response spike before the winner neuron spiked (false positive). This decrease prevents grids from being sensitive to irrelevant stimuli, not useful to distinguish the different classes as it has just lead to a wrong prediction. If at a certain position, C1 has emitted some spikes after the winner neuron spiked (false negative), then the weights are increased $A_p^+ > 0$ so that the neuron becomes sensitive to the a different pattern, which might be correct for the associated class. In addition, in case of punishment or silence (Equation 128), all weights in grids associated to the correct class are increased by a small amount ϵ so that class grids are more sensitive to this stimulus for the next predictions.

For the two datasets tested, the receptive field was selected to be the smallest region which can contain the entire object, more information on the reasons behind this can be found in the experiments section.

6.2.5 *An RL approach?*

It is questionable whether this approach actually belongs reinforcement learning. One of the main arguments against this question is the fact that the reward is constructed based on the training samples and is thus just a way of supervision. However, this approach is not only able to learn with training samples. If we do not have class labels, we can force the reward to be always positive, this would enable only the reward part of the method which behaves just like STDP, which is an unsupervised learning rule. Moreover, just like in the RL paradigm the agent is able to learn from its environment, the learning rule is capable of adapting itself to the given labels, even if they change during the training procedure.

If we consider a definition of RL that establishes that an agent learns by interacting with an environment and selects its actions essentially on trial and error using feedback from its own actions and experiences to maximise a notion of reward, the approach perfectly meets the definition. In fact, the class-specific patterns are created in a trial and error way by reinforcing weights which lead to a correct prediction. By acting this way we are actually maximising the reward (since we are reinforcing correct predictions and a positive reward is given only in case of a correct prediction).

6.2.6 *Experiments and results*

In order to test the efficacy of the proposed method, we decided to test it with the N-MNIST [34] (cfr. Section D.1) and POKER-DVS [63] (cfr. Section D.2) datasets. The first dataset shows digits moving in a 34x34 pixel array obtained through saccades while the latter features pips of cards moving on the screen.

The number of grids is chosen according to how diverse the classes are, e.g., if we want to classify digits from 0-9 in two classes: even and odd, we would need at least 5 grids per class. In this particular case, since each number corresponds to a different class, we can select 1 grid per class. Additionally we can increase the number of grids per class to give the model opportunity to recognise variations of the same class. In our experiments we set 2 grids per class. The parameters of the MSD were obtained through tuning, and were the ones which obtained the most stable pseudo-frame out of the events.

The neuron threshold parameter θ was also obtained through tuning and mostly showed positive correlation on the size of the receptive field, the bigger the receptive field, the bigger the threshold should be in order to avoid activations with only parts of patterns. It should not be too high either, so as to allow activation of patterns that are not necessarily composed of many spikes.

Similarly, the R-STDP parameters were also obtained through tuning. Their influence in convergence is still not fully understood since the method has no mathematical proof of convergence or guarantees. Roughly speaking, a_r^+ should be positive and should appropriately reward the spikes which help to a correct classification, in this case it the magnitude was selected higher than a_p^-, the weight decrease on punishment. If we had selected them equal in magnitude such as in the original R-STDP formulation by Mozafari et al. [65], in multi-class classification scenario, the knowledge obtained on a good prediction might be forgotten due to false positives during training.

The dropout rate was selected to yield good result during training. The decay is necessary in order to avoid the deactivation of learnt patterns during prediction. While in the first epochs there are still not many learned patterns, in successive epochs each grid per class might have a slight variation of a number, if we kept the dropout quite high, the pattern which best fits the data might be deactivated for the sample and possibly the other slight variation might be activated and updated, therefore "polluting" the existing pattern, or even causing the prediction to be inaccurate.

In particular the parameters used are given in Table 3.

Parameter	Description	Value
`grids_per_class`	Number of grids associated per class	2
θ_{MSD}	MSD threshold	150
τ_{MSD}	MSD exponential decay time constant	0.02 ms
`rf`	S2 receptive field	11
θ_n	neuron threshold	30
a_r^+	R-STDP, weight reinforcement on reward	0.075
a_r^-	R-STDP, weight decrease on reward	-0.025
a_p^+	R-STDP, weight reinforcement on punishment	0.005
a_p^-	R-STDP, weight decrease on punishment	-0.04
ϵ	R-STDP, weight increase for class-associated grids	0.004
`dropout`	Grid dropout rate	0.25
`dropout_decay`	Grid dropout epoch decay	0.7

Table 3: Parameters used for the N-MNIST dataset

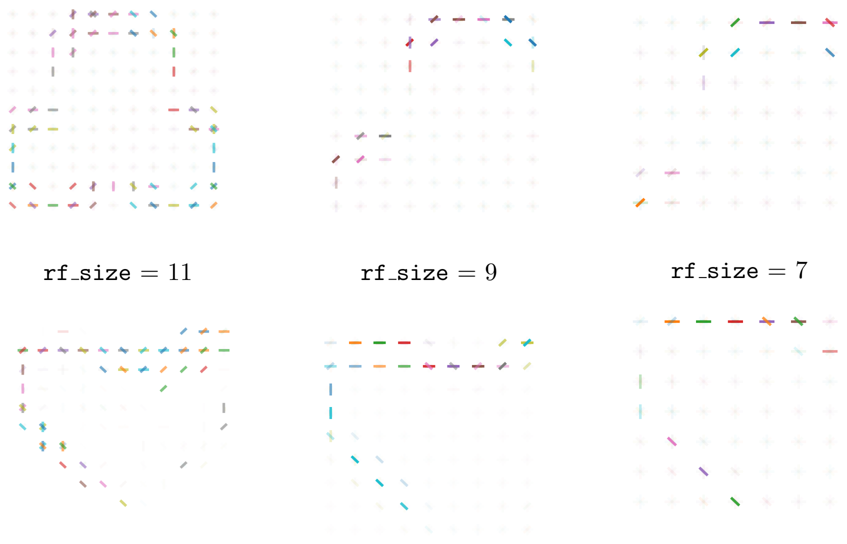


Figure 49: Class-specific pattern comparison with different receptive field sizes on the POKER-DVS dataset. The learning rule was able to find the class-specific patterns shown above for receptive fields 11x11, 9x9, and 7x7 while it was unable to find them with a 5x5 receptive field.

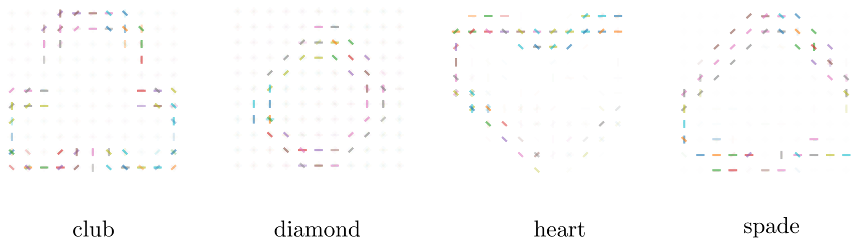


Figure 50: Learnt grid prototypes on the POKER-DVS dataset.

For the two datasets tested, the receptive field was selected to be the smallest region which can contain the entire object. By selecting a too small receptive field, the algorithm is likely to fail identifying a pattern that is able to properly separate the classes. For instance, a 5x5 receptive field might identify a semicircle which looks like an horizontally flipped C. Since this pattern occurs in numbers: zero, two, three and five, it will not be helpful for differentiating these classes. Even though it is possible to identify class-specific patterns which are not the size of the object as shown in Figure 49, the experiments revealed that a bigger receptive field size is more robust and achieves better accuracy scores, moreover it has some extra advantages that will be explored in Section 6.2.7.

The experiments revealed very interesting results, the method was able to learn the proper weights that represent a whole class. In Fig-

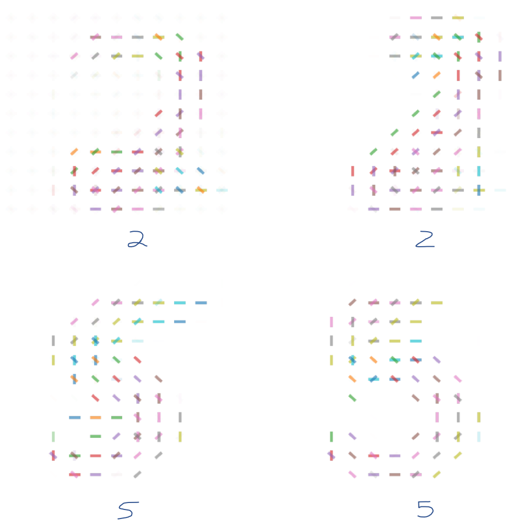


Figure 51: Example of two different prototypes of two digits learnt during the training mechanism. The approach is able to learn prototypes that represent the class, but also is able to learn different representations of the same class as it can be observed from the illustrated representations. Top: the 2 on the left includes a loop while the second is more straight and Z-shaped. Bottom: the 5 on the left is more S-shaped and the 5 on the right is more rounded on the middle-lower part.

ure 50, it is possible to observe the weights learnt during training. In order to make the weights more interpretable, a line in the direction of the associated Gabor filter is plotted with an opacity proportional to the weight value. Colour has no particular meaning.

By increasing the number of grids associated to each class, we can end up with different prototypes for a single class (Figure 51). To do so, it is necessary to use dropout in the training process, otherwise only one grid per class will be activated and react before the others.

Approach	POKER-DVS	N-MNIST	Useful for
Zhao et al. [39]	93.00%	86.60%	Classification
BOE [37]	93.00%	70.43%	Classification
HFIRST [36]	94.00%	71.15%	Classification
HOTS [62]	97.50%	80.8%	Classification
MUST [20]	99.00%	89.7%	Classification
REMUS (our work)	99.09%	84.4% ± 1.5% *	Classification & Detection

Table 4: Classification performance of different approaches.

* due to the complexity of the neuron simulations the algorithm was only run on a subset of the original dataset. Variations are due to the different train-test split. Algorithm took 6 hrs to run 8 training epochs on 300 samples.

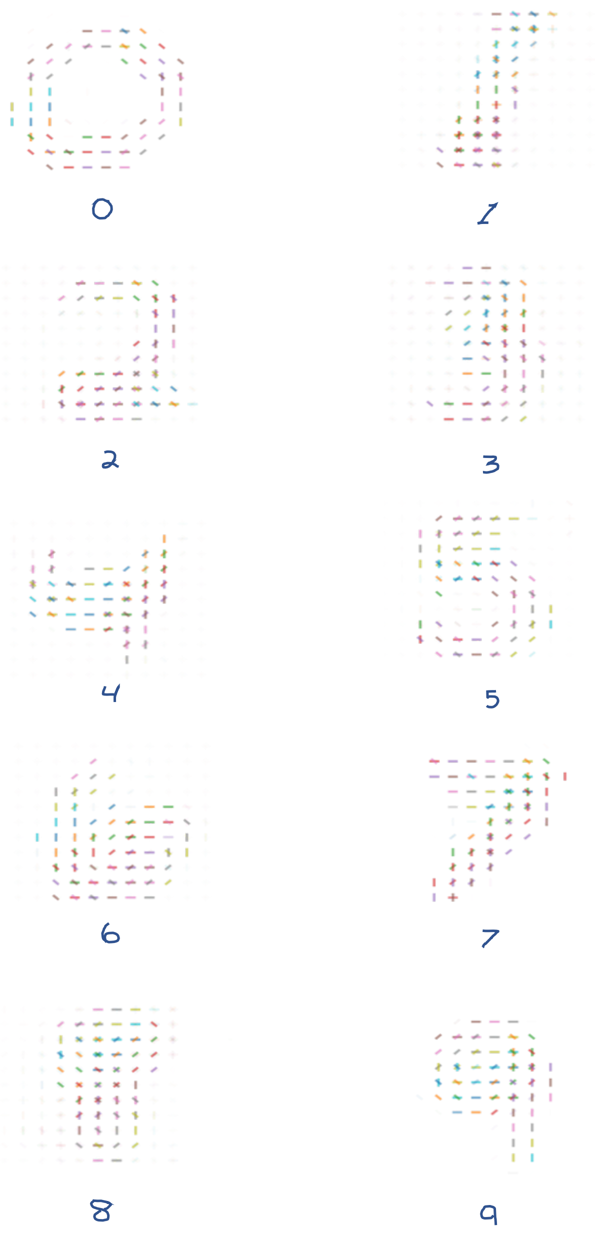
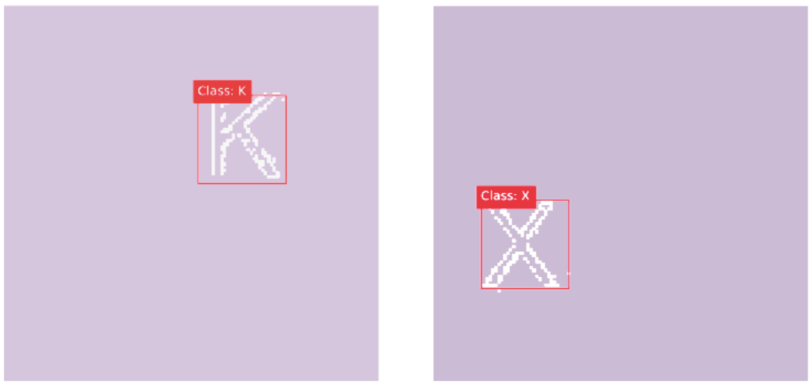


Figure 52: Illustration of the learnt weights with one grid per class of the N-MNIST dataset. As it can be easily observed, the weights perfectly represent a generic shape of each number.

Approach	Year	Kind	Feature Extraction	Event Processing (FE)	What is trained	Learning Method	Processing for Prediction	Classifier Input	Classifier
Masquelier and Thorpe [64]	2007	Visual Features	S1-C1-S2-C2, 1 scale, 4 orientations	None: natural images	Synaptic weights	STDP	None: natural images	C2 Potentials	OVA RBF
Chen et al. [67],	2012	Visual Features	S1-C1, 6 scales, 4 orientations	Event-driven	Tagged library of line features	-	HES: Temporal difference images	Line features	NN (Hausdorff-distance)
Zhao et al. [39]	2014	Visual Features	S1-C1, 4 scales, 4 orientations	SES: peak based	Synaptic weights	Tempotron	SES: peak-based	Feature I2L	SNN (MajVoting)
HFirst [36]	2015	Visual Features	S1-C1-S2, 1 scales, 12 orientations	Event-driven	Synaptic weights	Spike counts	Event-driven	S2 spikes	SNN (HFR)
BOE [37]	2016	Statistical Features	Popularity and specificity metrics	SES: threshold based	Specialty weight matrix	-	HES: Temporal difference images	Segment	Linear SVC or KNN
HOTS [62]	2016	Spatio-Temporal Features	Hierarchy of Time Surfaces with different time and spatial scales	Event-driven	Time Surface Prototypes	Clustering	Event-driven	Feature histogram	NN (Euclidean or Bhattacharyya distance)
R-STDP [65]	2018	Visual Features	S1-C1-S2-C2, 1 scale, 4 orientations	None: natural images	Synaptic weights	R-STDP	None: Natural Images	Feature I2L spikes	SNN (F2F)
MuST [20]	2020	Visual Features	S1-C1-S2-C2, 4 scales, 4 orientations	SES: peak based	Synaptic weights	STDP	SES: peak-based	Pattern activations	SNN (HFR)
ReMuS (this work)	2020	Visual Features	S1-C1-S2-C2, 4 scales, 4 orientations	SES: threshold based	Synaptic weights	R-STDP^{+}	SES: threshold-based	Feature log I2L spikes	SNN (F2F)



(a) Detection of letter K. (b) Detection of letter X.

Figure 53: Object detection with ReMuS. The approach is not only capable of correctly classifying very similar objects such as letters K and X, but it is also capable of perfectly finding the location in the scene.

6.2.7 *Going beyond classification: object detection*

Object detection is a problem which is more complex than classification [3]. The idea behind object detection is to identify an object in a scene (in the sense of finding its location within the scene), and get its class. Even though this may seem a different kind of problem than the one tackled, the proposed approach can be thought as a sliding-window object detection approach with a fixed square size (which equals to the size of the receptive field of the neurons). This is due to the fact that S2 neurons are capable of identifying a class pattern along all the input region thanks to their weight sharing mechanism. By keeping track of the first neuron inside the grid to spike, and appropriately choosing a receptive field which is big enough to contain a class object, it is possible to transform this object recognition approach into an object detection approach. Moreover, with an implementation architecture that exploits parallelism it is possible to reduce the computational cost.

In order to prove the added capabilities of this new method, we perform some experiments in the DVS-BARREL dataset [36] (cfr. Section D.3).

By selecting an S2 neural receptive field of 15 which roughly covers the size of the letters and digits in the dataset of `30x30` pixels (since the `15x15` neurons map to `2x2` max-pooled regions), it is possible to accurately draw a bounding box around the object in the scene while performing classification at the same time (Figure 53).

3 Object detection is a well studied problem in frame-based data and standard deep neural networks. One of the approaches worth mentioning is You Only Look Once (YOLO) [68] which tackles the problem from a regression perspective where the neural network predicts bounding boxes and class probabilities directly from full images in one evaluation.

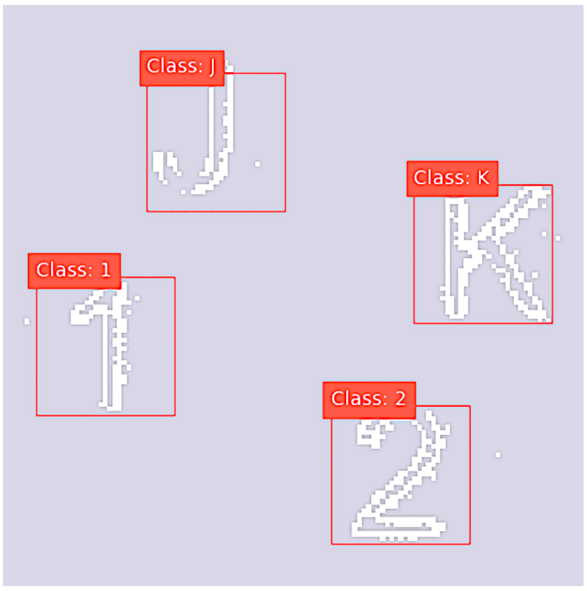


Figure 54: Multi-object detection on the original DVS BARREL recording.

Note that the algorithm, as it has been explained to this point, is only capable of identifying one object in the scene and not multiple objects, to remove this limitation it would be enough to allow multiple temporal winners during the prediction phase and prevent neurons to fire in regions where an object has already been identified.

The importance of setting a good parameter θ, which is the neuron threshold, becomes more important in this context as we are more prone to have false predictions. Figure 54 shows the detection of multiple objects in the scene.

Notwithstanding the powerful capabilities of this approach, it has the disadvantage of using the receptive field of a fixed size, making it difficult to draw accurate bounding boxes in digits or letters which are smaller than the receptive field. For bigger digits we can use spatial pyramids and work with subsampled pseudo-images.

With tested parameters of $\theta = 35$ and $\mathrm{rf} = 15$ with all the other parameters unchanged w.r.t. previous experiments, the algorithm is capable of producing, considering the event segmentation of samples, about 100 predictions out of a 0.12s sample, which potentially could be about 845 predictions per second.[4]

4 Note that this number just indicates the number of potential predictions given the Soft Event Segmentation (SES) method, under the assumption that the whole processing pipeline manages to act real time. Under computer simulations, prediction for a 0.12s sample requires about 26s on a MacbookPro 2016 dual-core, 8GB RAM.

6.2.8 *Going faster: event-driven prediction*

Until now, we have used a SES approach to segment events and get pseudo-frames due to the complexity of software simulations, however, it is possible to take inspiration from the event-driven architecture from [39] (EDHMAX, cfr. Section 6.1.1) to make the approach event-driven. To do so it would be enough to introduce a linear forgetting mechanism on Gabor filters and use a continuous event representation like leaky surfaces, such as the time surface representation [62] (HOTS, cfr. Section 6.1.4) in which the events are processed one-by-one and the current pseudo-image decays exponentially with a time constant τ.

6.2.9 *Discussion*

Event-driven approaches are relatively easy to implement with the current implementation, but have not been explored due to computational reasons. The ReMuS algorithm was written in Python (which does not enable very fast execution), and it uses software simulated neurons. Due to the custom-made learning rule and in order to properly debug the approach, software neurons were not written in simulators such as NEST or BRIAN, but instead were written in Python with some efficient libraries like `numba`, however there were not any sort of tricks like C-callbacks that could make it faster.

An implementation in neuromorphic hardware would solve the problems that arise with the simulations and open the door to this sort of approaches, which ideally would operate on the microsecond range. By implementing such an approach on neuromorphic hardware with silicon neurons, there is no need to loop over events, and over the exisiting neuron grids one by one like on software simulated neurons, instead, neuromorphic hardware works almost instant due to the fact that it operates with circuits and electrical currents instead of software simulations.

We would like to remark that this approach as well as previous work, cannot compete at the moment with traditional methods, we invite the curious reader to take a look at Appendix C to see non-bioinspired methods. The objective of the research is thus to conceptually create new models and approaches which lay out the foundations of future approaches and implementations. Only when neuromorphic hardware for SNNs becomes more widely available and less expensive, is when we will be able to fully take advantage of the power efficiency and asynchronicity which they offer.

To better understand the approach it is suggested to watch the following explanatory video: `https://youtu.be/AidIxJUFX_I`

7 ROBOT CONTROL

Now that we have explored the applications in feature extraction and object recognition, it is time to concentrate on probably what is the application which is best suited for the trinity of bio-inspired approaches: robot control.

In Section 7.1, we will briefly cover the evolution of robot control approaches using bio-inspired technologies. In Section 7.2, we introduce some scenarios created to show the difference in performance of standard RL robot control approaches and bio-inspired RL robot control approaches when using data from event-based sensors, and introduce the agents used from a high-level point of view. Finally in Section 7.3 we implement the scenarios and the agents in a 3D world simulator and compare their performances.

7.1 CONTEXT AND STATE OF THE ART

Applications of SNN for robot control date back to the early 2000's where several researchers tried to use them for autonomous navigation. Floreano and Mattiussi used evolutionary methods for vision-based SNN controllers that performed navigation tasks in different contexts: a small two-wheeled robot navigating in a rectangular arena with textured walls [69], and blimps and micro-flyers navigating in an indoor scenario [70]. Di Paolo [71] performed a first attempt to control a robot coupling STDP with an evolutionary strategy, and Florian [72] showed that networks using STDP evolved faster (in terms of generations) than networks with static synapses. Hagras et al. [73] used an adaptive crossover and mutation genetic algorithm on a robot with nine ultrasound sensors and four bump sensors, to converge faster to a solution that exhibited the desired edge-following behaviour. In 2009, Wang et al. [74] built a robot that is capable of following a wall by using an SNN controller and 16 evenly distributed ultrasonic sensors, even when obstacles are present.

Despite all these successful experiments made on robot control, at this point, none of the researchers had used event-based camera vision sensors. As shown in previous sections, event-based cameras, also called AER vision sensors, offer considerable advantages over standard frame-based cameras, namely a high dynamic range, avoidance of motion blur, efficiency by avoiding redundancy, and a very low latency (microsecond range). These characteristics and the fact that they can be processed by SNNs makes them a perfect fit for ap-

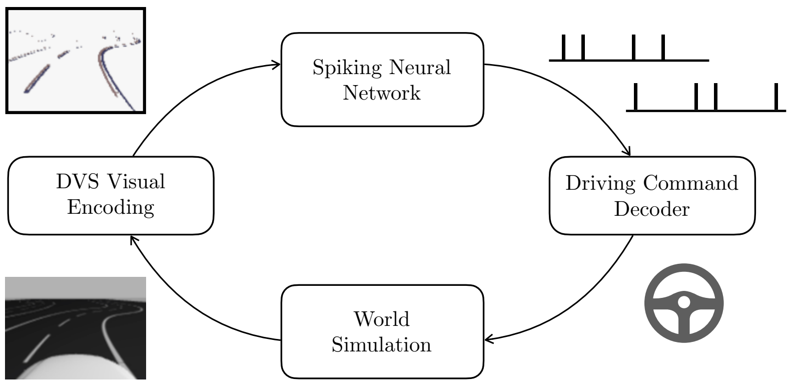


Figure 55: The general simulation framework for robot control.

plication needing efficient and fast real-time computations such as autonomous robots and vehicles, regardless of their field.

In 2013, Perez-Peña et al. [75] used DVS data to reproduce intended movements performed by humans with a neuroinspired algorithm (SVITE: Spike-based VITE). Some years later, in 2016, Moeys et al. [76] used a CNN with data from a DVS to control a "predator" and follow another which acted as a "prey", while in 2017, Blum et al. [77] implemented a spike-based robotic controller on neuromorphic hardware that is able to perform reactive obstacle avoidance and target acquisition in an unknown environment using as sensory input just a DVS and an IMU.

Lately, there has been a lot of interest in RL and it seems natural to try to apply such mechanisms also for robot control. Vasilaki et al. [52] used a SNN to control a robot in the Morris water navigation task, a very famous task used to study spatial learning and memory which consists in finding an invisible or visible platform that allows the agent to escape the water. Evans [78] used R-STDP on a robot with range and touch sensors that had to learn the correct behaviour to collect food items (avoid poison, empty containers, etc.), while some other researchers like Rosenfeld et al. [79] attempted neuromorphic control using policy gradient-based algorithms for SNNs.

It was not until Kaiser et al. [80] developed a framework to evaluate neural self-driving vehicle applications that approaches combining SNN, RL and AER sensors really began to emerge. This framework, shown in Figure 55, consists in using the Robot Operating System (ROS) as a communication middleware between a world simulator like Gazebo or V-REP which represent both the environment and the agent, and an SNN that controls the agent. In such framework, the simulation data is passed to the SNN who then takes the actions needed (such as steering commands), and sends them to the simulated vehicle.

The vehicle in the framework is a Braitenberg-like vehicle with two wheels and a simulated DVS camera. The event-based camera is obtained by subtracting consecutive frames of a video stream and keeping pixels which have a difference (positive or negative) with an absolute value greater than a given threshold. This loses the asynchronicity of the sensor but allows a close simulation of this novel visual sensor. The vehicle has two motor neurons, whose activity is used to control the steering angle of wheels of the vehicle with muscle modelling.

These motor neurons also set the speed. The speed is inversely proportional to the steering angle and keeps history of the last commands in order to make smooth transitions.

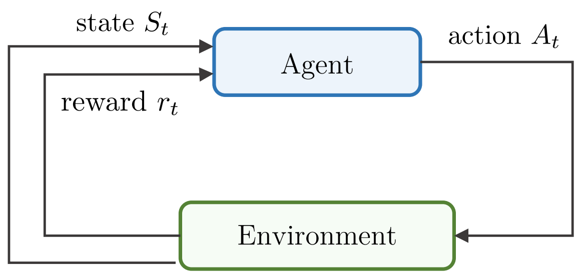
This framework inspired Bing et al. [81] to improve the original paper baseline. Instead of picking a set with handpicked-weights, the authors translated policies learned with standard Deep Q-Learning to an SNN and explored SNN learning rules such as R-STDP (Section 5.2.2). With the same inspiration, Bing et al. also explored R-STDP on a snake-like robot used to perform target tracking tasks. Such experiments inspired in turn Tieck et al. [82] to use R-STDP on a robotic arm that needs to reach a target and to perform manipulation tasks.

The simulation framework has shown how it is possible to structure simulations with these technologies. It is therefore highly relevant to create some new scenarios and test whether novel approaches using R-STDP are in effect better than standard RL approaches as there has been very few research on this topic.

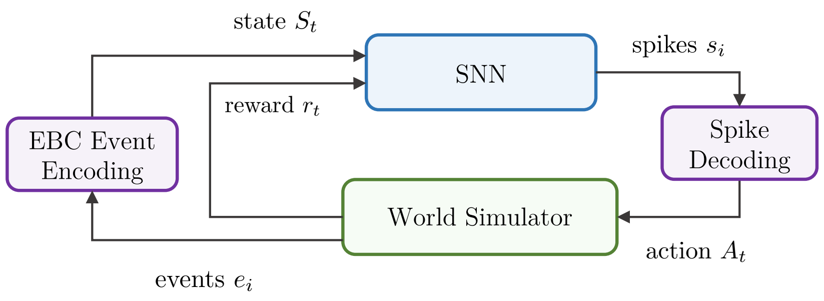
7.2 UAV ROBOT CONTROL

In order to demonstrate the usefulness of R-STDP in robot control, and its superiority with respect to traditional reinforcement learning algorithms, we developed for this thesis two scenarios that need to be accomplished by an Unmanned Air Vehicle (UAV), in particular by a Quadricopter, having as a constraint the use of data gathered from an event-based camera as an only input for the algorithms.

From a high-level overview, the simulations are based on the self-driving vehicle simulation framework (cfr. Figure 55), which behaves similarly to the standard RL framework as seen in Figure 56.



(a) The standard RL framework.



(b) The high level simulation framework for an SNN agent.

Figure 56: RL Frameworks for standard and SNN agents.

The tasks to be accomplished by the UAV are the following:

① A ball-following task. The objective of this task is to train a UAV controller to keep a moving ball always in the camera's field of view. The idea of this task is to replicate the working mechanism of "follow-me" drones that are able to keep track of target-object (generally the user, in outdoor sports settings) despite erratic and high-speed movements. In this case, we chose the target object to be a ball, which during the simulations follows a pre-defined path that is unknown to the UAV.

② A line-following task. The objective of this task is to follow a line painted on the floor. The drone moves forward at a constant speed and the controller must learn when to move left or right and rotate left or rotate right in order to follow the line as precise as possible (centered and at a 90 degrees orientation).

These tasks are not particularly hard to solve, in fact, both tasks can be solved easily with human knowledge of the task and by the assignment of static weights to the SNNs. The idea of these scenarios is thus to demonstrate the learning capabilities of the approaches to be tested, to see whether they are able to outperform the human solution, to understand how long they need to be trained, and how the approaches are ranked among themselves.

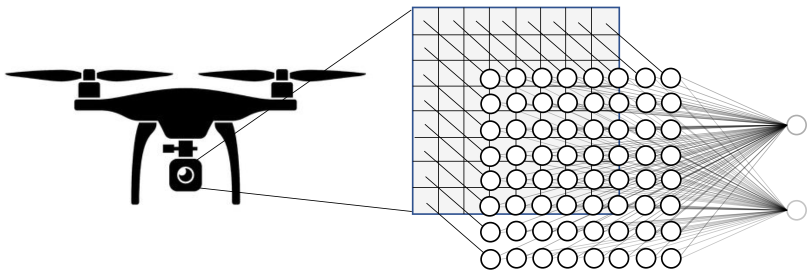


Figure 57: SNN agent network overview.

7.2.1 *The standard* RL *agents*

The standard reinforcement learning agents will be Deep RL agents that take a vector as input, and output an integer (in case of discrete-action compatible approaches like DQN), or a floating point vector (in case of continuous-action compatible approaches like DDPG).

In order to create the input vector, we will not create a full image out of the events as it would make the input dimension too big and possibly too complex to learn a policy, also the use of CNN is superfluous as the state is not complex enough to require their use and it would only make the training longer and will make the prediction phase slower. Instead, the input DVS grid will be partitioned in some regions, each region will measure whether there is an event during that step and use as an input a vector of 1s and 0s containing whether the regions have experienced an event or not during the simulation step.

7.2.2 *The* SNN *agent*

The SNN that will be used for the simulations will transform the events received into a particular event representation. The input DVS grid will be partitioned in some regions, and each region will count the number of events that have occurred in that simulation step. Each of the regions will be associated to a neuron, that will encode the event count using a *rate coding* mechanism [1] and generate a certain spike train. These neurons will be connected in an all-to-all manner to a set of output neurons as shown in Figure 57. The synapses connecting the neurons associated to the DVS regions and the output neurons have synaptic weights which are trained with the R-STDP/dopamine-modulated learning rule (Section 5.2.2).

1 Since we already lack the temporal aspect of events due to the nature of simulation, the best model that encodes the intensity of events in a certain region is the Poisson model which maps to rate coding.

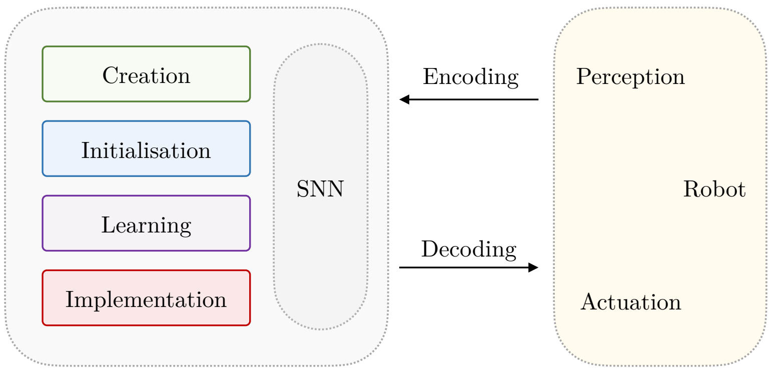


Figure 58: The general design framework for robot control developed by Meschede et al. [53].

The general design framework used for the SNN agent was the one suggested by Meschede et al. [53] and shown in Figure 58. This framework consists on a four-step process that starts by determining the network model and determining the appropriate neuron model for each layer (in this case feedforward and IF neurons). The second step consists in choosing an appropriate learning algorithm and a proper initialisation for the neurons in the layers (R-STDP and fixed initialisation with a weight value at the middle of the allowed value range [min, max]). The third step is performing the learning on the network by properly showing the target to the network, propagating an error or reward signal and reset the environment in episodic tasks. While the last step consists in validation of the network in a pre-defined scenario and exploration and optimisation in unknown scenarios.

Of course we also need to find a way of representing information of the environment (be it physical or simulated) in the activity of a neuron (*neural encoding*), and a mechanism that interprets neuron activity and translates that activity into electrical signals that drive actuators (*neural decoding*). This was briefly covered in Section 2.3. In our case, we use a rate coding mechanism and a muscle modelling decoding which will be explained into details in Section 7.3.5.

7.3 EXPERIMENTS

Before delving deeper in the scenarios and the intricacies of the algorithms, we briefly introduce the different software and libraries used for the simulation, explain what they are, how they work, and how they can all be linked together so that their interaction results in simulations that are as lifelike as possible.

The technologies used for the simulation were not arbitrarily chosen, instead, they were based on the simulation framework developed

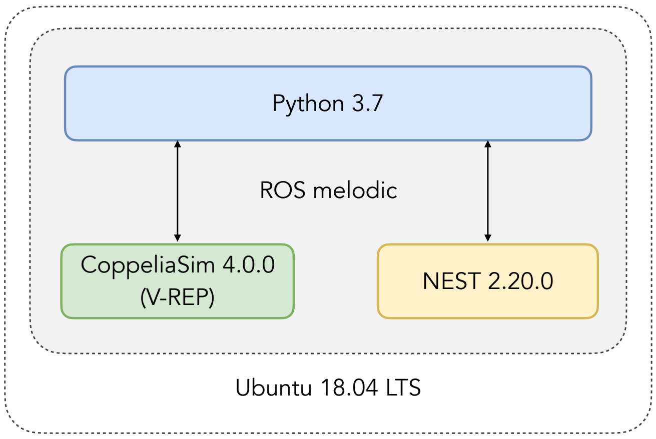


Figure 59: Software stack overview.

by Kaiser et al. [80] and on previous work. To be precise, the software used for the simulation is described in Figure 59.

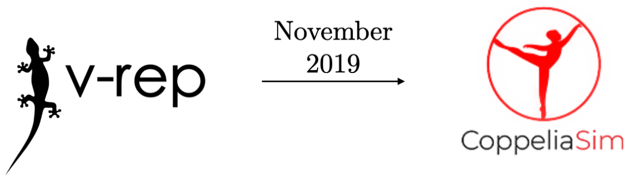
Each piece of software is responsible for some very specific task in the simulation. All the algorithms were developed in Python 3.7 and can be run from a GUI, a Jupyter notebook, or a simple script. CoppeliaSim is a robot simulator and is explained in more details in Section 7.3.1. The Robot Operating System (ROS) manages communication between the simulator and the algorithms and is explained in Section 7.3.3, while NEST is an SNN simulator and is explained in Section 7.3.4. In this same section, we explain the implementation of the SNN explained from a high-level view in Figure 57.

In Section 7.3.5, we describe how the output spikes are encoded to an action. In Section 7.3.6, we explicitly mention the simplifications used in the simulation scenarios. In Section 7.3.7, we introduce a layer of abstraction used to communicate with the world simulator. Finally in Section 7.3.8, we show the results for the different agents implemented.

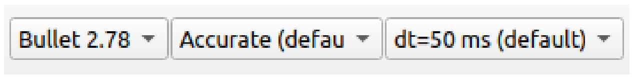
7.3.1 *The simulator: CoppeliaSim*

CoppeliaSim, previously known as Virtual Robot Experimentation Platform (V-REP), is a powerful cross-platform robot simulator software developed by Coppelia Robotics. It is a complete 3D robot simulation software, with an integrated development environment that allows to model, edit, program and simulate almost every aspect of any robot or robotic system (e.g. sensors, dynamics, properties, etc.). It is an alternative to Gazebo, the robotic simulator by the Open Source Robotics Foundation (OSRF).

Each simulation in CoppeliaSim has a certain temporal resolution, accurateness and Physics Engine among Bullet, ODE, Newton, and

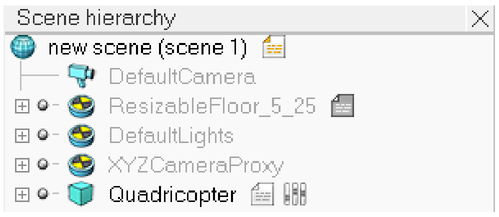


(a) CoppeliaSim was born as a fork of the V-REP project which offered more features and some improvements that make it run faster with respect to V-REP. The transition occurred on the release of v4.0.0 and all the previous code is 100% compatible.



Bullet 2.78 | Accurate (defau | dt=50 ms (default)

(b) Default parameters for simulation and the ones used on this work.



Scene hierarchy
new scene (scene 1)
DefaultCamera
ResizableFloor_5_25
DefaultLights
XYZCameraProxy
Quadricopter

(c) A simple scene with a quadricopter. Objects like floor, cameras and lights are added by default when a new scene is created.

Figure 60: CoppeliaSim

Vortex, which determine the general parameters of the simulation. The parameters used for the simulations are the default ones which are shown in Figure 60b.

7.3.1.1 *Scenes*

Inside the simulator, we can create a *scene* file (`.ttt` file extension) which includes the main elements that allow us to appreciate a simulation: the floor, the objects which are part of the simulation, a light that illuminates the scene, some cameras that allow us to view the scene, some pages with views that make it easier to visualise relevant information, graphs, and so on. An example is shown in Figure 60c. We create a scene for each of the two tasks.

7.3.1.2 *Behaviour*

The objects inside a scene can have a child-script which specifies their behaviour. This script represents a small collection of functions and routines written in a a not so popular, but powerful, efficient, lightweight, embeddable, multi-paradigm programming language called Lua.

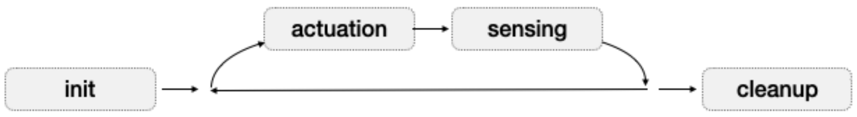


Figure 61: Note that the actuation phase comes before the sensing phase as performing actions in a reverse order might end up with an incoherent display of an object. For example an object could have measured 10m to the wall but then moved 1m towards the wall, at the end of the loop we will have 10m sensed but will see only 9m of distance to the wall.

Through this script we define 4 fundamental callback functions which are useful to describe the behaviour of each of the objects in the simulation: the drone, the DVS camera, the ball, etc. These 4 callback functions are:

- `sysCall_init()`, this callback function will be executed just one time when the child script is called for the first time. The call to the child script is not necessarily at the beginning of a simulation, this is due to the fact that scripts can be called or added in the middle of a simulation. In case of complex models like a drone, this part of the simulation is responsible of getting *handles* to sensors or actuators and specifying constants. Handles are integer numbers that work as an internal reference pointer for an object:

  ```
  sensorDVS128=sim.getObjectHandle("DVS128")
  ```

  once we get the handle by getting it with the object name, we can call methods on the handle as it was the object itself.

- `sysCall_actuation()`, this callback function will be executed in each simulation step, during the actuation phase of a simulation step. It is responsible of handling the dynamics and kinematics of the object.

- `sysCall_sensing()`, this callback function will will be executed as well in each simulation step, but in this case during the actuation phase of a simulation step. The code is thus in charge of handling all the functionality regarding sensors like proximity sensors, and collision detection.

- `sysCall_cleanup()`, this callback function is executed just one time before a simulation ends. The code is usually in charge of restoring object's initial state, configuration and parameters.

Note that there is a very specific order in which callbacks are executed, this order is shown in Figure 61.

7.3.1.3 *The Scenarios*

BALL-FOLLOWING SCENARIO

In the Ball-following scenario shown in Figure 62 and Figure 63, a blue ball/sphere follows a Bezier curve defined by 8 control points which are positioned in a four-branch star shape. The goal of this scenario is to move the position of the drone so that the ball is always located at the center of the DVS sensor in a similar way to how a "follow-me" drone operates.

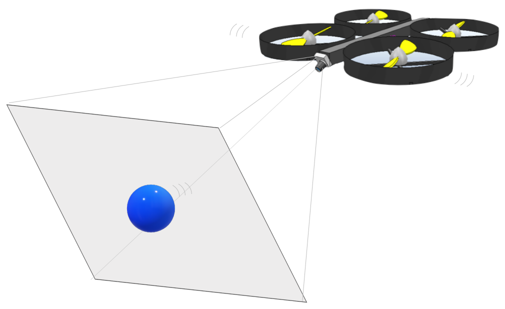


Figure 62: Overview of the ball-following task.

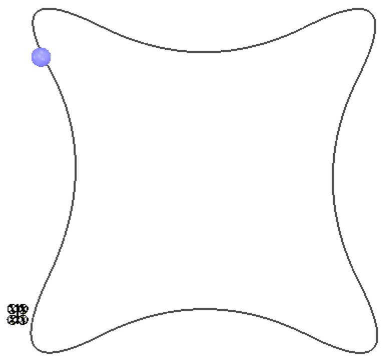


Figure 63: The ball moves at a speed of 0.075 m/s along a path that is invisible to the drone. The altitude of the drone is fixed at 2.4m from the ground. The episode finished whenever the ball is not in the field of view or when a lap to the path has been completed.

Line-following scenario

In the line-following scenario shown in Figure 64 and Figure 65, the drone must follow an hexagon-shaped line and move itself in a way that the line is always kept at the center and in the best orientation possible (i.e. 90 degrees when there is a straight line). The drone moves at a constant speed and chooses whether to go left or go right and/or rotate at each timestep.

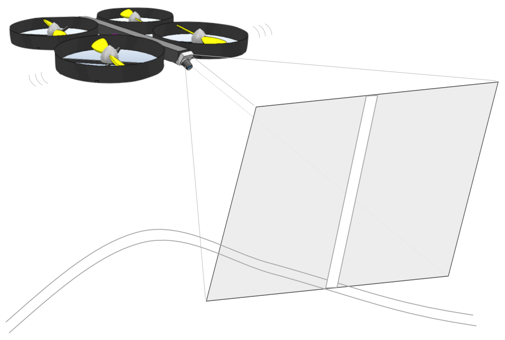


Figure 64: Overview of the line-following task.

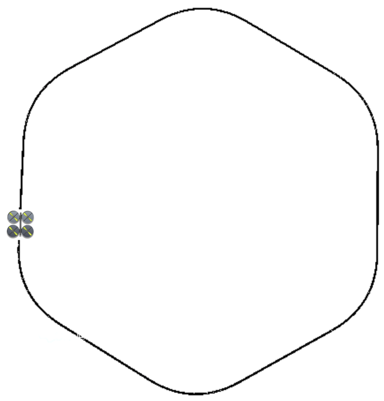
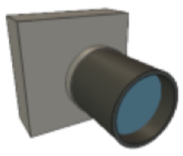
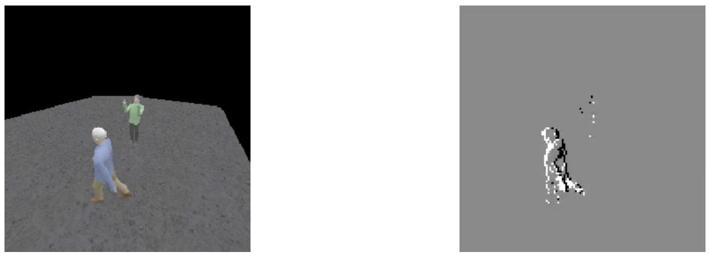


Figure 65: The drone moves forward at a speed of 0.075 m/s. The altitude of the drone is fixed at 2.4m from the ground. The reason why the line is shaped like an hexagon is due to the fact that the curves created by such shape have an angle greater than 90 degrees which allows the estimation of orientation of the line in the scene. This is needed to assign a reward. In scenarios with different shapes, the drone must be close enough to the line so as to only see one line and not two like in a very arched curve. The episode finishes when the line is not in the drone's field of view or a lap has been completed.



(a) The simulated model of a DVS camera has by default a "128x128" resolution, a perspective lens angle of 65 degrees and a near/far clipping plane of (0.05, 2) meters.



(b) Left: a standard vision sensor simulation on CoppeliaSim. Right: the DVS sensor simulation of the same scene with positive events represented as white pixels and negative events represented as black pixels.

Figure 66: Simulated DVS camera model

7.3.1.4 *The Event-based camera*

Since July 8th, 2013, CoppeliaSim has a Dynamic Vision Sensor (DVS) model as a courtesy of IniLabs, which is a spin-off company of the Institute of Neuroinformatics at the University of Zurich and ETH Zurich (Tobi Delbruck et al.). The DVS camera is located under:

```
models/components/sensors/DVS128.ttm
```

Some simulations performed with this sensor can be seen in Figure 66b.

7.3.1.5 *The Drone*

The drone used for the simulations is provided by courtesy of Eric Rohmer and is located under:

```
models/robots/mobile/quadricopter.ttm
```

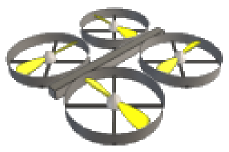


Figure 67: The model of the drone has 4 propellers in an H frame.

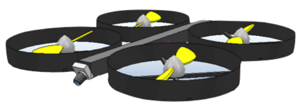


Figure 68: The UAV model used for the simulations.

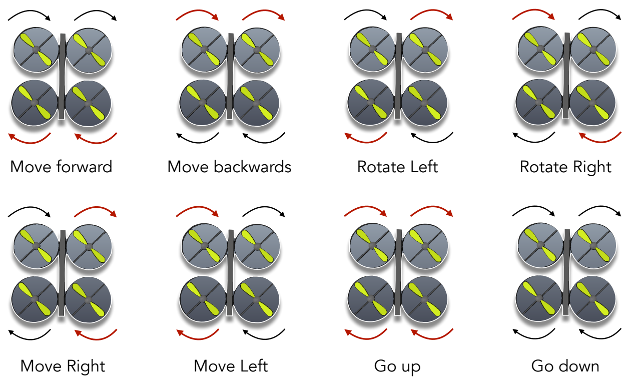


Figure 69: Movements of a quadricopter. Red arrows indicate a higher propeller speed.

7.3.2 *The UAV*

The Unmanned Air Vehicle (UAV) in the simulations consists in the quadricopter drone described in Section 7.3.1.5 which has attached a DVS camera sensor described in Section 7.3.1.4. The camera is placed on the front of the quadricopter, on the supporting bar, with an inclination of 30 degrees as shown in Figure 68.

In order to induce directed motion in a drone, one must control very carefully the speeds of each one of the propellers. This is shown in Figure 69. But in order to simplify the scenarios, we do not control manually each of the propellers, but instead the movement commands are given at a high level (move right, move forward, rotate right, or move 1.4 units to the right/left, etc.). It is the internal PID controller provided with the model that ensures a smooth transition and control of each of the forces of the propeller. Note that this simplification has multiple purposes. First and foremost, it avoids a lot of training since there is no need to learn the speeds for correctly hovering the drone and the associations needed to create a certain movement, and secondly it avoids abrupt movements as there is a smooth transition towards the target. Thanks to this simplifications the learning task can focus on which actions to take, instead of needing to learn how to perform the actions.

As a further simplification for all scenarios, the z (vertical) coordinate of the drone is fixed, leaving as only degrees of freedom movements relative to the drone's x and y axis, and the rotation α (normal to the z axis).

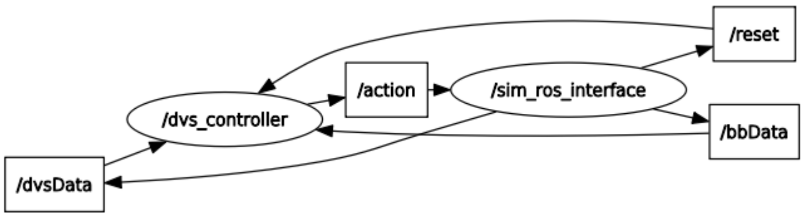


Figure 70: This is the graph created by the command `rqt_graph` in a ball-following simulation. It is the easiest way to visualise the publish-subscribe relationships between ROS nodes (ellipses).

7.3.3 *ROS*

The Robot Operating System (ROS) is an open-source, meta-operating system for robots which provides several services including: low-level device control, message-passing between processes, code writing, code building, and package management, etc.

Among its many goals, ROS intends to help roboticists to design complex software by providing an abstraction of small and independent software programs called *nodes*. These nodes communicate with one another and do so with what is called ROS master or `roscore`. Once this master node is active, it is possible to run other nodes (programs that use ROS).

The primary mechanism that ROS nodes uses to communicate is to send messages which are organised into *topics*. These topics act with a standard publish/subscribe mechanism and have a specific message type.

As it can be observed in Figure 70, in order to receive and send messages for the simulations, we need a couple of ROS nodes: one in the CoppeliaSim simulator which is created by the plugin called `ROSInterface` and another which is created in Python, and which in the example above takes the name of `dvs_controller`.

As stated before, each message has a type which contains the format of each one of the fields. It is possible to create custom messages, however, it is possible to use some of the already existing message packages provided by ROS like: `geometry_msgs`, `sensor_msgs`, or `std_msgs`.

Example of a ROS message type:

```
geometry_msgs/Vector3
        float64 x
        float64 y
        float64 z
```

Figure 71: NEST simulator logo. The NEST simulator is maintained by the NEST Initiative who has advanced computational neuroscience since 2001 by pushing the limits of large-scale simulations of biologically realistic neuronal networks.

7.3.4 *NEST network*

The NEural Simulation Tool (NEST) [83] is a simulator for spiking neural network models which tries to follow the same logic of an electrophysiological experiment, however instead of performing the simulation in the physical world, the simulation takes place inside the computer.

This simulator makes it possible to run SNN simulations through high-level Python scripts. However, Python code allows for dynamic typing and is not compiled, introducing a significant overhead when running simulations and interpreting the code at runtime, for this reason, in order to obtain the best possible performance for the simulations, the simulator generates code from the user-written functions into an intermediate simulation language which is then run with an interpreter written in C++. This is done under the hood and it is completely transparent to the end user. Moreover, recent developments have significantly improved memory management and therefore reduced memory requirements. As a consequence of this highly efficient implementation, NEST has been able to simulate more than 1.73 billion neurons connected by 10.4 trillion synapses on a Japanese supercomputer [84] and allows thousands of neurons on a standard desktop computer.

```python
import nest
# Integrate and Fire Neurons
neuron1 = nest.Create("iaf_psc_alpha")
neuron2 = nest.Create("iaf_psc_alpha")
voltmeter = nest.Create("voltmeter")
# Initial state properties
nest.SetStatus(neuron1, {"I_e": 1000.0})
# Synapses
nest.Connect(neuron1, neuron2,
             syn_spec={'weight': 20, 'delay': 1.0})
nest.Connect(voltmeter, neuron2)
# Simulate Network
nest.Simulate(100.0)
```

Code Snippet 1: A simple NEST simulation with two IF neurons.

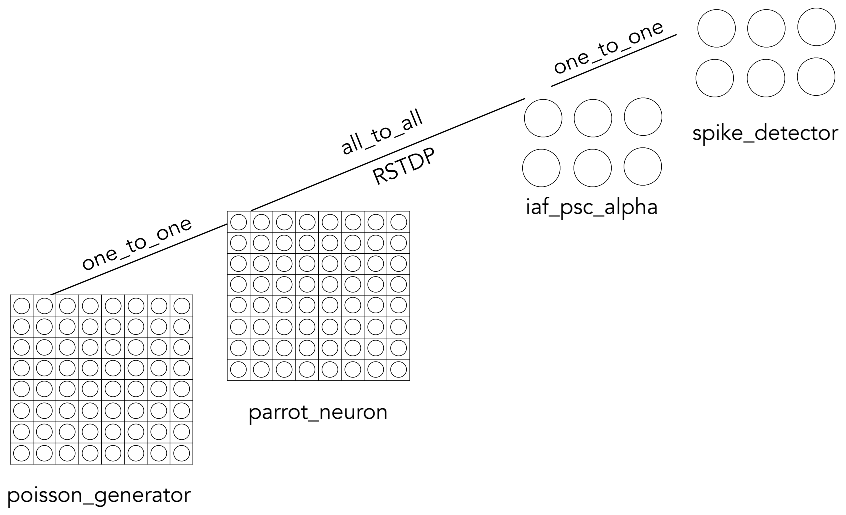


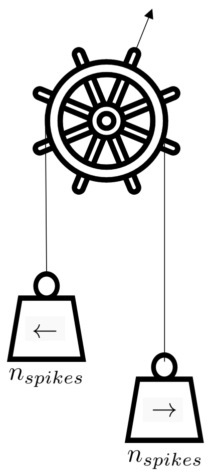
Figure 72: NEST implementation overview of the SNN network used for both scenarios. From a high level overview, the network simply consists in an input layer which is based on 64 partitioned regions of the DVS pixel array performing event count which are connected in an all-to-all manner with `stdp_dopa_synapse` synapses to a set of output neurons.

NEST is not the only SNN simulator, indeed there are other simulators such as Nengo, Brian, and NEURON, however, what makes NEST very useful is the fact that it provides a large set of tested neurons and synapses. In particular, for the two scenarios, we will use the `stdp_dopa_synapse`[2] which implements the R-STDP algorithm proposed by Izhikevich [50] which was described in Section 5.2.2.

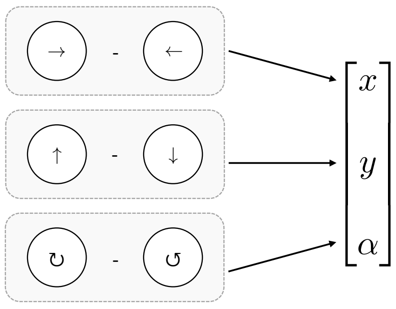
The network used for both scenarios is shown in Figure 72. We use a pair of neurons in the output layer for each of the degrees of freedom which will be used to power the action model explained in Section 7.3.5, and the input layer is a set of Poisson neurons, this is because the event count is transformed into a rate for the neurons and set at each simulation step.

The way that NEST handles output from Poisson neurons [3] is that it generates a different spike train for each of the receiving (post-synaptic) neurons, we want to avoid this as each output neurons would see a different input spike and may behave differently in case

2 `stdp_dopa`: nest-simulator.readthedocs.io/en/nest-2.20.1/models/stdp.html
3 Note that by using Poisson neurons implies we use rate coding instead of temporal coding. Even though temporal coding has been shown to be more informative than rate coding [85], in this case, the information encoded to spikes lacks by the temporal aspect. We use rate coding since the event count is used to generate a spike train and we do not care so much about the actual instant at which spikes are sent but the number of spikes per unit of time (rate) which is proportional to the event count value.



(a) Useful representation of the muscle modelling decoding mechanism, highlighting that the importance lies in the delta between spikes associated to the neurons.



(b) Drone continuous action vector.

Figure 73: The action model for the SNN agent.

of leakage. In order to avoid this, we use what is called Parrot Neurons, which merely repeat the same spike train to all post-synaptic IF neurons. Finally, we use a spike detector to measure the number of output spikes from the output layer.

7.3.5 *The action model*

In case of a continuous action set, the action model contains a pair of neurons in the output layer for each of the 3 degrees of freedom. The bio-inspired UAV agent uses the spikes of an SNN to take actions. The action is performed with a muscle modelling decoding based on the amount of spikes emitted for each of the neurons. This approach is given by the following formula:

$$\frac{n_{spikes}(positive) - n_{spikes}(negative)}{n_{max\_spikes}}, \tag{129}$$

where positive and negative indicate the positive and negative direction for each of the degrees of freedom (e.g. for the x axis, right is positive and left is negative), while the max number of spikes depends on the simulation time and the refractory time of the neurons. As we can observe from Figure 73a, it is not the actual number of spikes that matter, but the delta between the spikes of the neurons relevant to a degree of freedom.

In a discrete action environment, we use actions with some kind of number association (one-hot encoded in the approach). For example in the BALL-FOLLOWING scenario, we use the following actions:

① $\leftarrow$ - go left

② $\rightarrow$ - go right

③ $\uparrow$ - go forward

④ $\downarrow$ - go backwards

⑨ $\varnothing$ - do nothing

⑤ $\nearrow$ - go northeast

⑥ $\nwarrow$ - go northwest

⑦ $\searrow$ - go southeast

⑧ $\swarrow$ - go southwest

Note that the diagonal actions are needed as even though the same result can be achieved by a combination of lateral and vertical movements, such movement is impossible to achieve in a single step, which is possible in the continuous scenario.

In a very similar way, we define the actions for the LINE-FOLLOWING scenario, just that in this case, since the drone constantly moves forward, we would only have lateral and rotation movements and their combinations as available actions.

7.3.6 *The simplifications*

In order to make the simulations tractable, we performed two main simplifications:

1. In the ball-target scenario, the background was set as non-renderable so it does not generate any events. This simplification is needed to reduce the computational power required, the background generates many noisy events and makes the simulation stutter. This simplification can be solved in a real-life implementation by filtering noise events. In Appendix B, we propose a new kind of noise filter that addresses this limitation and is even helpful for some object recognition approaches.

2. We deliberately restrict the drone from rotating in the BALL-FOLLOWING scenario, even if in real life a drone is able to rotate, by letting the drone rotate, we make the control problem an ill-posed problem since there are many solutions. This multiple solutions add complexity to the training phase which may elevate training hours to an unfeasible quantity. The only degrees or freedom for this scenario are thus movements along its x and y axis.

7.3.7 *The environment*

In order to provide a layer of abstraction on the robotic simulator, all the logic involving the simulator: scenes, object interaction, dynamics, reward and so on was encapsulated in a Python file called `environment.py`. The idea of this layer of abstraction is to provide an environment which behaves almost identically to an OpenAI Gym Environment [86] on which standard RL-algorithms are normally used and tested.

```
env = CoppeliaSimEnvironment()

for episode in range(n_episodes):
  state = env.reset()
  done = False
  episode_reward = 0

  while not done:
    action = agent.choose_action(state)
    new_state, reward, done, info = env.step(action)
    episode_reward += reward
    # some learning mechanism on the agent ...
```

Code Snippet 2: This generic "while" loop abstracts all the working mechanisms of the simulator. The only part left to implement is an agent which chooses the action to take at the next step of the simulation.

Internally, the `environment.py` file has a hook to the CoppeliaSim simulator achieved through a remoteAPI set of files and configuration. The hook allows the Python script to start and stop the simulation as well as to dynamically load the scene.

```
/
└── remote_api
    ├── simConst.py
    ├── sim.py
    └── remoteApi.so
```

Figure 74: In order to execute actions like loading a scene, starting and stopping the simulation, these 3 files are needed. The `simConst.py` file contains constants among which there are constants defining command return codes. The `sim.py` file is a Python wrapper which calls C-functions contained in the `remoteApi.so` library file provided with the installation package. Note that due to the compiled nature of C, this last file is OS-dependent, for MacOS it has a `.dylib` extension, while for Windows it has a `.dll` extension.

As we just mentioned, all the complexity of the simulator is abstracted in an `environment.py` file. However, it is now time to explore how this file handles multiple things among which is the model of the state, the communication with the simulator for receiving and sending data through ROS, the computation of the reward, and so on.

Each `environment.py` file is responsible of performing the following tasks:

1. Creating the needed ROS publishers and subscribers.
   In order to receive data from the simulation and to send the actions that the drone needs to perform in the next simulation step, the environment script is responsible of subscribing to at least 2 topics (one to get the DVS data and another determining when to stop the simulation), and of publishing to one topic the action to take.

2. Specifying the callback functions.
   Each of the ROS subscriptions needs to specify a callback function which implements the logic to be executed when a message of that topic is received.

```
self.dvs_sub = rospy.Subscriber('dvsData',
                                 Int32MultiArray,
                                 self.dvs_callback)

self.reset_sub = rospy.Subscriber('reset',
                                   Bool,
                                   self.reset_callback)

self.action_pub = rospy.Publisher('action',
                                   Float32MultiArray,
                                   queue_size=1)
```

Code Snippet 3: In both scenarios, each environment has at least 2 subscribers and one publisher. The first subscriber listens to the topic where the DVS events are sent while the second subscriber signals whether the scenario has ended (reached a terminal state). The publisher on the other hand is responsible for sending the action. Note that the type of message depends on whether the environment is using a discrete action set (that would just use an Int) or a continuous action set (e.g. a Float for each propeller or direction).

3. Computing the reward.
   Once an action has been executed, the file should compute the reward by using the state and some additional information. This

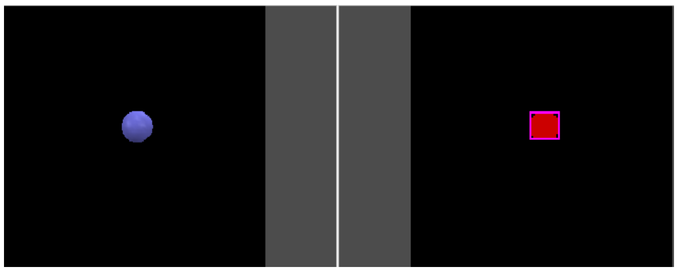


Figure 75: Left: an image of a ball in a scene without any object detection algorithm. Right: the bounding box found with the OpenCV function in CoppeliaSim.

additional information can be obtained by using a new subscriber which provides this additional information.

A) *Ball-following task.*

In the case of the ball-following task, the wrapper uses an additional subscriber which receives data from a standard image sensor. This standard image sensor performs object detection on the current image using the OpenCV library [87] and draws a bounding box around the target as shown in Figure 75. The scope of this additional image sensor is to understand the exact location of the object in the scene and how far the object is from the camera sensor. Note that this wrapper is used only for computing a reward. Once the training phase is over, the drone does not need at all such sensor to choose the action to take at a given state.

Since the aim of this scenario is to keep a ball at the center of the DVS image sensor, for the reward computation we consider as an ideal situation an instant where the center of the ball's bounding box obtained from the additional subscriber is at coordinates `63x63px` which is the center of the DVS Sensor, and the height of the bounding box is `13px` tall. This last condition is to avoid being too close or too far from the target, since otherwise it is possible to keep the ball at the center regardless of the what is the distance to the ball.

The reward is computed differently for traditional RL - algorithms and for SNN algorithms. The reason of this difference is due to the fact that the SNN algorithm requires the assignment of a reward for each of the neurons involved in the network deciding the action taken. The reward is defined linearly dependent on the robot's distance to the ball center. Neurons associated to different directions of the same axis receive the same reward but opposite in sign.

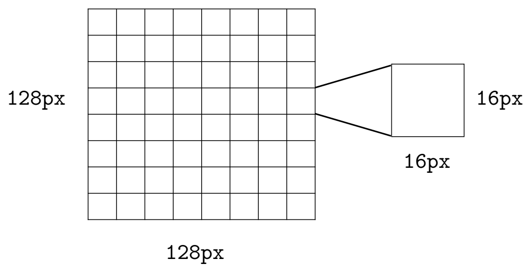


Figure 76: DVS sensor partitioned into `16x16px` regions. The state is thus represented by a vector of 64 integers, where each integer represents the event count inside a region in the grid, independent of the polarity.

B) *Line-following task.*
Since the objective of the line-following task is that the drone follows a line, the reward computation considers as ideal a situation in which the angle of the line in 90 degrees vertical and the line is located at the center of the image sensor (pixel `63`). In order to compute the position and the angle of the line, the wrapper uses an additional subscriber which receives data from a standard image sensor which performs canny edge detection and Hough Lines Transform on the current image using the OpenCV library.

④ Taking a step.
Once the agent has given an action, the wrapper should publish the action to the simulator, wait for the new data collected after executing the simulation. Use the callback functions to adjust the current state and return a new state, a reward, a termination boolean (the episode is terminated when the line or the target ball is not within the field of view of the DVS sensor, or when a lap is completed), and some additional information for metrics. Inside this function, there is a very important command:

```
rate.sleep()
```

which basically allows the step loop to run at nearly (best effort) the exact rate (in Hz) that has been specified while initialising the rate object (e.g. `rate = rospy.Rate(50)`).

⑤ Computing the new state.
The event camera outputs events. In order to transform the data received from the simulation to something that resemble more a standard state that can be used by algorithms, we decided to partition the `128x128` pixel grid into an `8x8` grid where each cell counts all the events happening in its `16x16` region of interest as shown in Figure 76.

7.3.8 *Experiment results*

As it was explained in the last section, thanks to the environment abstraction, all we need to do is to implement an agent which encapsulates the logic for taking actions in the environment. The agents implemented for the ball-following and the line-following task were the following:

- A DQN agent (explained in Section 4.6.5).
- An SNN agent using synapses with static weights set with human knowledge of the problem.
- An SNN agent using R-STDP synapses for learning.
- A DDPG agent (explained in Section 4.7.3).

BALL-TARGET *scenario*

In order to demonstrate the difference in performance between the several available methods, we compute the average euclidean error/distance to the target. Since we want the ball centered, what we would like is thus to have the center of the ball positioned close to the middle of the DVS pixel grid.

To understand also a bit more the performance of the algorithms, we plot the density heatmap of the ball-positions, with a darker colour to indicate that such positions have been visited a lot of times during simulations, and with lighter colour to indicate very few simulation steps in which the ball was located in such positions. The absence of colour (paper colour/white), indicates that such positions were not visited at all.

On the sides of the density plot we can find the marginal distributions. These distributions describe the projected position of the ball from the point of view of only one axis.

In Figure 77, we compare the aforementioned density heatmap of the ball-positions for the algorithms that converged to a solution. Unfortunately the DDPG did not converge to a solution after more than 24 hr of training.

In Figure 78 and Figure 79, we show how the error and convergence varies with the training episodes for SNN-R-STDP and respectively, DQN. The approach using an SNN with static weights is not shown since it is not trained and converges from the very beginning, while DDPG is not shown since it did not converge.

Notice that the chosen metric of euclidean distance is more informative than the average position of the ball. As it can be observed from the density of the simulation with fixed weights, the average position obtained by the approach would be still very close to the center, but this is only due to the fact that positive and negative errors will compensate each other. By using the euclidean distance to the target (x_t, y_t), with the formula

$$\text{Error} = \frac{1}{T} \sum_{i=1}^{T} \sqrt{(x_i - x_t)^2 + (y_i - y_t)^2}\ , \tag{130}$$

such errors do not compensate each other because the error is always positive.

As it can be observed from the density plots (Figure 77), even though DQN has the peak of its distribution on the desired position, it does does not always succeed on keeping the ball at the center, the visited positions cover almost all the pixel grid and the wide distribution indicates a lower precision. This density plot represents the result achieved after 14 hrs of training. Maybe with more time it could have reach a slightly better solution but no drastic improvement is expected.

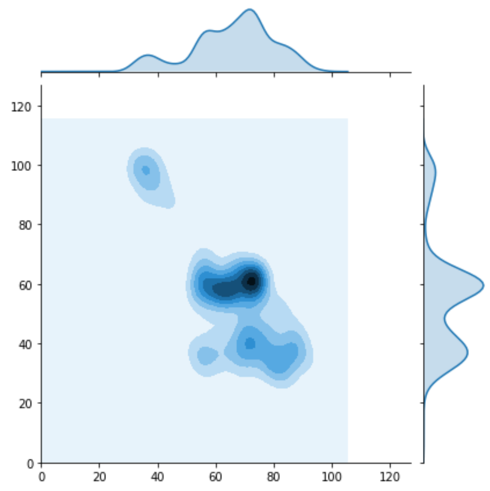
If we observe the density plot with the fixed weights, we see a kind of loop around the center, which achieves a kind of wobbly and wide plateau on the marginal distribution. On the other hand, the SNN approach with R-STDP achieves a nearly gaussian distribution which is peaked at exactly the desired position and does not visit other nearby states as much as in the case with static weight or DQN.

In all the simulations, we can see that on all the density plots, the upper left side contains some visited states, this is due to the initialization. In fact the ball is not positioned at the center when the episode begins, which causes these inevitable not centered positions.

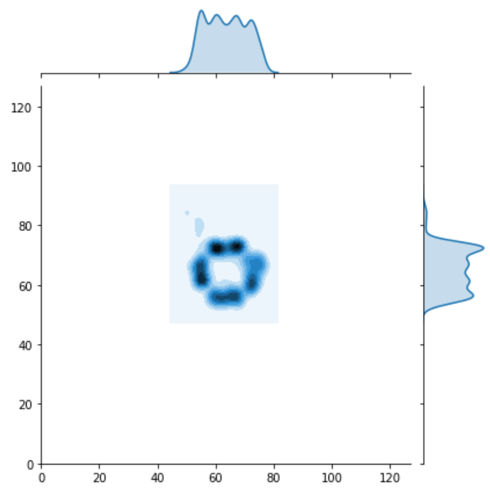
Approach	Avg Pos.	Avg. Error	Steps to convergence
Static	(61.61, 63.66)	10.017	0 (fixed)
SNN + Static Init	(60.74, 63.69)	8.55	0 (fixed)
SNN	(61.2, 62.62)	6.77	60000
DQN	(62.88, 65.17)	21.48	696000

Table 5: Performance on target-following scenarios. Note that since static weights do not need training and already encode a human solution for the problem, the drone converges from the very beginning. In this table we include an extra case in which the SNN uses the static weights as initialisation, while this performs better than just using static weights, the initialisation heavily conditions the solution and underperforms w.r.t. a solution found with equal weights initialisation.

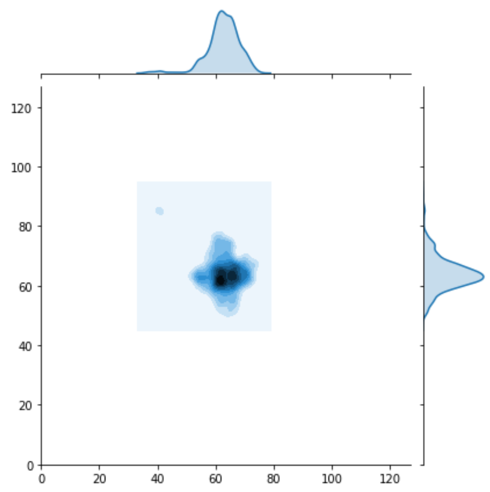
Video of the SNN agent in a BALL-TARGET scenario: https://youtu.be/64-rlS3Ne1Y



(a) DQN. Corresponding to an average euclidean error to the target of 21.48.

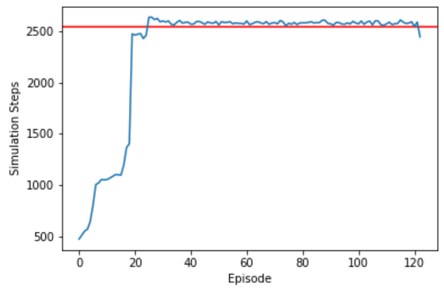


(b) SNN + fixed weights. Corresponding to an average euclidean error to the target of 10.017

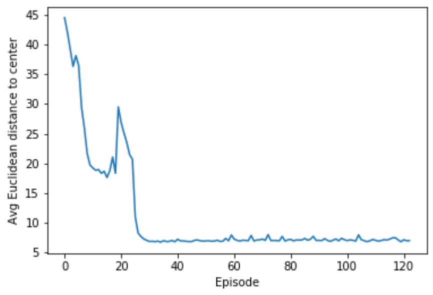


(c) SNN + R-STDP. Corresponding to an average euclidean error to the target of 6.77.

Figure 77: In figures (a)-(c), it is possible to appreciate the density function of the ball positions during an episode. It is worth noticing that SNN + R-STDP outperforms the other methods.



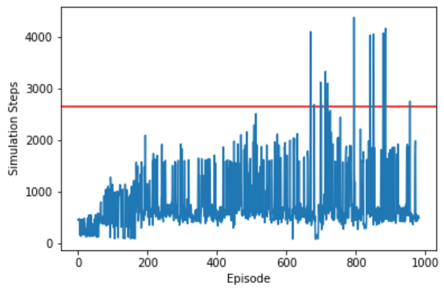
(a) After about 30 episodes ($\simeq$ 60000 simulation steps), the algorithm converged to a solution which finished the episode at every time.



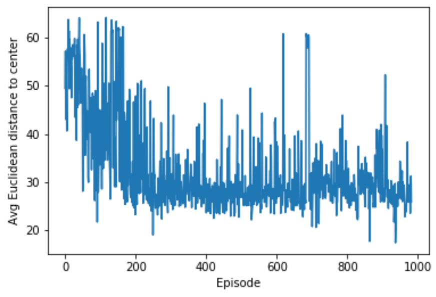
(b) Graph of the average euclidean distance to the center on an episode.

Figure 78: Results for the SNN-R-STDP agent.

Episodes with steps above the red bar are considered completed. The number of steps in a completed simulation are not fixed, they might slightly vary from run to run. This is due to the simulated nature of the experiments.

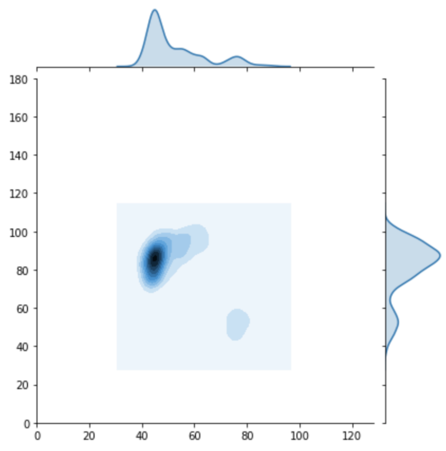


(a) After about 650 episodes ($\simeq$ 696000 simulation steps), the algorithm successfully concluded an episode.

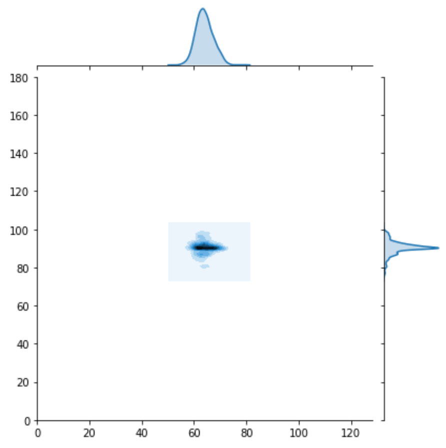


(b) Graph of the average euclidean distance to the center on an episode.

Figure 79: Results for the DQN agent.



(a) Density plot for a DQN approach.



(b) Density plot for an SNN-R-STDP approach.

Figure 80: Results for the line-following scenario.

Line-following *scenario*

In the line following scenario, we aim to have a line centered and with an angle of 90 degrees vertical. To show the difference in performance between the agents, also in this case we generate density plots. In this scenario, we plot the mean position of the center of the line on one axis and the orientation on the other one, the objective would be to have the density concentrated on the middle point of the density plot which is (63, 90). The results are shown in Figure 80 and Table 6.

Also in this scenario DDPG failed to converge while SNN-R-STDP achieved the best result. Choosing static weights revealed itself more challenging than expected. While most of the times rotation is enough to complete such a famous scenario, in this case the drone's rotation works fundamentally different from a two-wheeled vehicle which usually is set to accomplish this task. Its rotation has a different pivot point, it usually uses two sensors and is much closer to the line to follow. Unfortunately despite several attempts no set of weights chosen was good enough to make the drone perform a lap around the line.

Approach	Avg Pos.	Episodes to convergence
SNN	(63.04, 88.47)	10
DQN	(54.28, 84.87)	648

Table 6: Results for the line-following scenario.

From the experiments we can conclude that a SNN outperforms classical Deep-RL agents for the two tasks and even outperforms a solution encoding previous knowledge. Even though this was proven in software simulations and in simplified scenarios, this is clearly a good sign for the years to come and for upcoming robot control approaches, this means that it is possible in theory to be more power efficient by using bio-inspired methods without making any compromise on the performance.

Video of the SNN agent in a Line-following scenario: `https://youtu.be/Oote4EmCDmM`

8 CONCLUSIONS AND FUTURE WORK

8.1 CONCLUSIONS

This thesis addressed the problem of whether reinforcement learning, spiking neural networks, and event-based cameras could be used together and in what way. As a result of the research done, this thesis provides three main contributions:

1. Filled a gap in the literature by giving a clear perspective on how the all three elements are linked and can be used together (Chapter 5).

2. Developed ReMuS, an innovative event-based object recognition approach based on SNN and a RL-inspired learning rule whose performance is comparable to state of the art approaches, and in some aspects outperforms current approaches (Chapter 6). To the extent of our knowledge, this is the first approach that combines the trio of bio-inspired approaches to perform object recognition. This novel approach opens the door to countless kind of variations and many potential future developments (Section 8.2).

3. An application of the three bio-inspired technologies in a novel control scenario of an UAV (Chapter 7), which showed through some simulations, that RL techniques relying on SNN perform better than traditional RL techniques, not only due to their efficiency advantages, but also from a performance point of view.

This thesis also provides three side contributions:

1. `aertb`, a PyPI Python library simplifying the processing of event-based data and providing some functions for common tasks (Appendix A).

2. TAR, a simple and efficient noise filter for event-based data which can be used as a preprocessing step for multiple approaches and different fields of application (Appendix B).

3. R-STDDP, a learning rule which translates R-STDP to a synaptic delay paradigm while keeping its properties, potentially opening doors to future approaches combining synaptic delay and synaptic weight learning rules (Section 8.2.1).

8.2 FUTURE WORK

8.2.1 RSTDDP

The Reward-modulated Spike-Timing Dependent Delay Plasticity (R-STDDP) learning rule is a novel learning rule for spiking neural networks developed during the thesis work based on R-STDP which instead of modifying synaptic weights, modifies synaptic delays in a RL-inspired manner.

Most SNN approaches modify the synaptic weights connecting pairs of neurons, however it is also possible to modify the synaptic delays of the neurons. A synaptic delay acts, just as its name indicates, by delaying any incoming spike by a time which is equal to its value.

The objective of R-STDDP is to make a correct class-specific neurons spike before other neurons. This is done by reducing the delay (i.e. setting it to zero) for important spikes contributing to the correct class decision, while increasing the delay to those who do not do so. Assuming we start with a positive, non-zero delay, we can summarise the learning rule as:

$$\Delta d_{ij} = \begin{cases} \mathit{reward} & \begin{cases} Ar^{-} \cdot d_{ij} \text{ if } t^{f}(j) \geqslant t_i \\ Ar^{+} \cdot d_{ij} \text{ if } t^{f}(j) < t_i \end{cases} \\ \\ \mathit{punishment} & \begin{cases} Ap^{-} \cdot d_{ij} \text{ if } t^{f}(j) \leqslant t_i \\ Ap^{+} \cdot d_{ij} \text{ if } t^{f}(j) > t_i \end{cases} \end{cases} . \quad (131)$$

To verify the efficacy of the learning rule, the spiking grid demonstrator contained in the original paper [65] was replicated. This learning rule potentially opens doors to future approaches combining synaptic delay and synaptic weight learning rules.

8.2.2 OBJECT RECOGNITION

One big disadvantage of the proposed approach for object recognition is the fact that, even though the approach works in practice and gives good results, there are still no mathematical proofs of convergence or any sort of guarantees for the learning rule. It would be very interesting to formally prove its convergence as well as to understand deeply how parameters have an impact in the results and the training phase and how their values can affect convergence.

Another disadvantage is that the approach uses events to generate a static frame. This practice is not ideal in the event-based paradigm

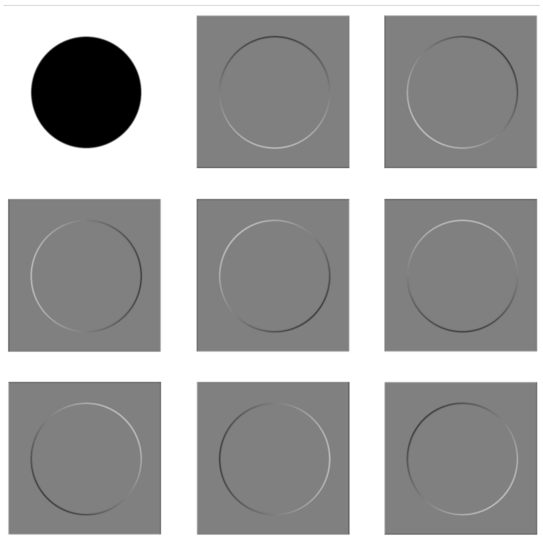


Figure 81: Sobel Filters computing the image gradient of a circle as if they were oriented every 45 degrees.

because it quantises event timestamps, discards temporal information, and the resulting images are highly sensitive to the number of events used. It would be interesting to explore event-driven variations of the algorithm which address these problems. An idea could be to use representations such as time surfaces instead of pseudo-images.

It would also be interesting to see whether additional image filters could be helpful for classification. For instance, directed Sobel filters likes the ones shown in Figure 81, could be really useful for the motion-like frames generated from a batch of events since they compute the gradient of the image.

It could be also interesting to see whether the receptive field could be decreased by using a multi-layered SNN that would be able to process the activation of S2 neurons and update the weights of the inner layers based on the final prediction with the same RL-reinforced rule.

As mentioned in the respective section, the proposed approach for object recognition learns some grid weights which then represent a whole class. Moreover, since the S2 neuron spiking in a grid has a well defined receptive field, the method could be seen as something beyond an object recognition approach. Since it not only recognises the class, but it also locates the position of the class in the frame, it closely resembles a fixed-size sliding window object detection approach where the size of the receptive field determines the size of the sliding window. As future work, it would be interesting to try whether non-squared filters can be used or whether there is a way to generate a better bounding box rather than just using the size of the receptive field.

Part III

APPENDIX

A AERTB

At the time this master thesis is being written, there are several manufacturers and research groups operating with event-based cameras: IniLabs (Tobi Delbruck), the IMSE Neuromorphic group (Bernabé Linares Barranco and Teresa Serrano Gotarredona), the Robotics and Perception Group at Zurich (Davide Scaramuzza et al.), Intel Labs (Garrick Orchard et al), and the French company Prophesee (Amos Sironi, Vincent Lepetit et al.), and some other groups around the world.

Unfortunately the field is not mature enough and as a consequence there are no common standards regarding data encoding formats, almost everyone has their own proprietary format and a different file extension. The variety of encoding formats heavily impacts researchers as publicly available datasets suffer from this different file formats and encodings, and a great amount of time needs to been lost to properly load the data into the programming language where a new algorithm is being developed.

To solve this problems, during the thesis work, a Python library was created to easily load these several file extensions and provide some useful utilities such as file format conversion to HDF5, creation of a gif based on the events, iterating an HDF5 file with specified groups, number of samples, etc. This library is available on the PyPI repository and installable with the command `pip3 install aertb` where aertb stands for AER-ToolBox.

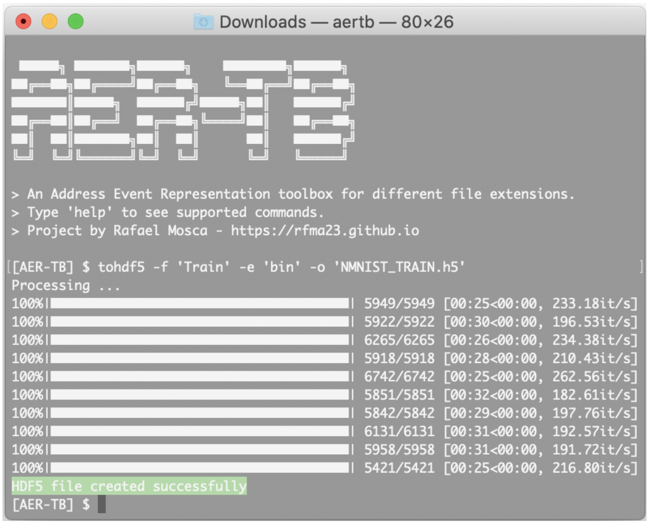
Its usage is quite simple and intuitive:

```python
from aertb.core import PolarityEventFile

f = PolarityEventFile('/folder/filename.dat')
events = f.load_events()
```

Code Snippet 4: Load operation with the `aertb` library. The events are returned as a recarray (record array) which is a highly efficient data structure provided by `numpy` which acts as a ndarray (n-dimensional array) that allows easy field access using attributes.

The library also offers a Command Line Interface (CLI) and a shell (Figure 82) for converting datasets to HDF5 and making gifs out of a sample without the need to create a python script and execute it. The shell is installed with the installation of the library and by just typing



```
> An Address Event Representation toolbox for different file extensions.
> Type 'help' to see supported commands.
> Project by Rafael Mosca - https://rfma23.github.io

[AER-TB] $ tohdf5 -f 'Train' -e 'bin' -o 'NMNIST_TRAIN.h5'
Processing ...
100%|████████████████████████████████████| 5949/5949 [00:25<00:00, 233.18it/s]
100%|████████████████████████████████████| 5922/5922 [00:30<00:00, 196.53it/s]
100%|████████████████████████████████████| 6265/6265 [00:26<00:00, 234.38it/s]
100%|████████████████████████████████████| 5918/5918 [00:28<00:00, 210.43it/s]
100%|████████████████████████████████████| 6742/6742 [00:25<00:00, 262.56it/s]
100%|████████████████████████████████████| 5851/5851 [00:32<00:00, 182.61it/s]
100%|████████████████████████████████████| 5842/5842 [00:29<00:00, 197.76it/s]
100%|████████████████████████████████████| 6131/6131 [00:31<00:00, 192.57it/s]
100%|████████████████████████████████████| 5958/5958 [00:31<00:00, 191.72it/s]
100%|████████████████████████████████████| 5421/5421 [00:25<00:00, 216.80it/s]
HDF5 file created successfully
[AER-TB] $
```

Figure 82: `aertb`: custom shell successfully executing a command which transforms a whole dataset made of several 'bin' files in a single file with an HDF5 format.

the command `aertb` from a normal terminal window, the shell utility will open. This shell is very convenient as it gives help and offers autocompletion for all the supported commands.

The Hierarchical Data Format version 5 (HDF5) is an open source file format that is able to manage large, complex and heterogeneous data. One of the main characteristics of this file is that it is fast, cross-platform, and allows easy sharing of data, a whole dataset containing many samples can be sent as a single file. This format has high level APIs with C, C++, Fortran 90, Java and Python.

```python
from aertb.core import HDF5File

# open the file and select the groups of interest
nmnist_train = HDF5File('folder/NMNIST_Train.hdf5',
                        groups=['0', '1'])

# Randomly select 20 samples on each of the groups
train_iter = nmnist_train.iterator(n_samples_group=20)

for sample in tqdm(train_iter):
        # we can access sample.name
        # we can access sample.label (group)
        # we can access sample.events
```

Code Snippet 5: Once a dataset has been converted to HDF5 we can easily access groups and events with the simple commands above.

B TAR

The approach proposed, that we like to call Temporal Active Regions (TAR) Filter is a very simple method for filtering noise events, it is based on a very simple assumption: valid events in the immediate future will happen near recent events. This assumption is also based on the very high temporal resolution of an event-camera, while very fast moving objects might appear to hop from frame to frame in conventional image sensors, in event-based cameras we can track movement very precisely, and movement of an object will be certainly captured by the sensor.

B.1 The algorithm

The precise working mechanism of this filtering approach is also simple. All we need is a mask with the same size as the camera sensor. At the arrival of an event, we signal activity in an R-neighbourhood of the event by increasing the value of the mask by one. The event is then filtered if the value in the mask associated to the pixel has a value under a threshold θ, otherwise the event is not filtered. Note that regardless of whether the event is filtered or not, the activity is signalled in the mask, this is to possibly learn new objects entering the scene, while at the very first events, the valid events might be considered noise, in a very near future we understand from the activity that we are not dealing with noise but a new object moving in the scene. Finally, to forget past events and focus more on recent events, we introduce an exponential decay. At the arrival of each event we decay the mask values with a time constant τ.

Algorithm 14: Temporal Activity Region (TAR)

```
initialise mask $M(x, y)$ $\forall x, y \in$ pixel grid, last_ts = 0
**foreach** *event* **do**
    Exponentially decay mask values
     $M(x, y) = M(x, y) * e^{-(event.ts - last\_ts)/\tau}$
    Increase activity in R-Neighborhood
     $M(x, y)[x - R, x + R][y - R, y + R] += 1$
    **if** $M(x, y) > \theta$ **then**
        keep
    **end**
    last_ts = event.ts
**end**
```

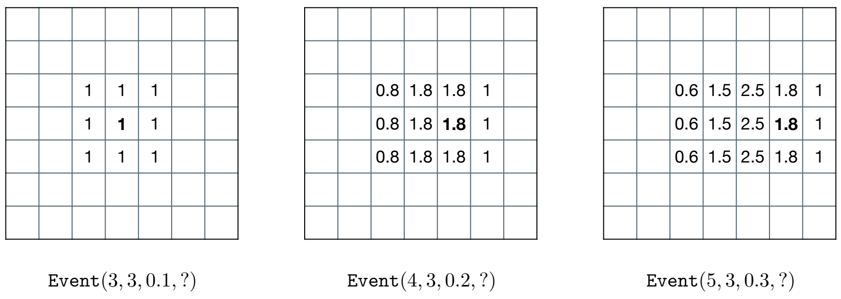


Figure 83: A simple example of the TAR algorithm on a 7x7 pixel grid, with $\tau = 0.5$ and $R = 1$. The question mark on the event's polarity highlights the fact that the algorithm process the events in the same way, irrespective of the event's polarity.

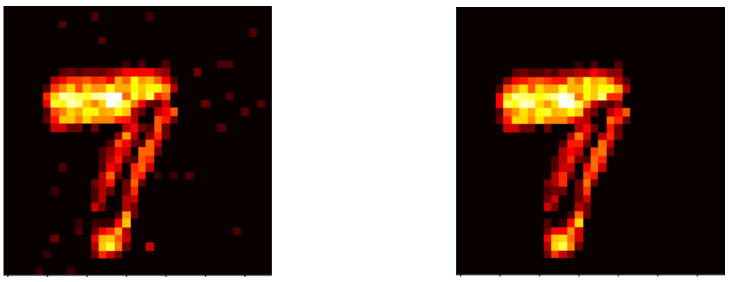


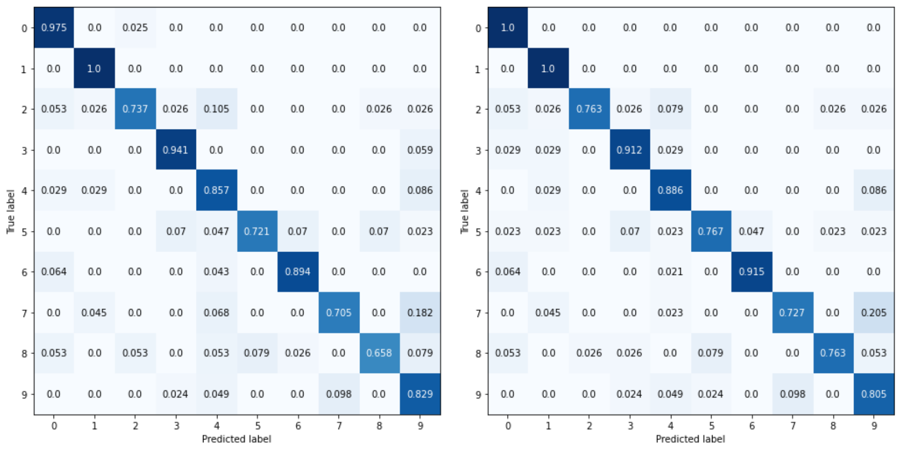
Figure 84: A digit from the N-MNIST dataset, the digit on the left contains non-filtered events while the image on the right contains only events filtered by the TAR method

To better understand the noise filtering algorithm, a simple example evidencing the working principle can be found in Figure 83, while in Figure 84 the effect of the noise filtering algorithm can be easily appreciated. An important thing to remark is that this algorithm does not distinguish the polarity of the events.

B.2 RESULTS

To verify the effectiveness of the approach we decided to test the impact of having TAR as a preprocessing step. In particular we decided to use BOE [37] (Section 6.1.3) as an algorithm on two datasets, N-MNIST [34] and Poker-DVS [63].

Some tests showed that with an appropriate choice of parameters (R, τ, θ) we could increase the accuracy score of the algorithm, being the sole difference the TAR preprocessing step.



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	0.975	0.0	0.025	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.053	0.026	0.737	0.026	0.105	0.0	0.0	0.0	0.026	0.026
3	0.0	0.0	0.0	0.941	0.0	0.0	0.0	0.0	0.0	0.059
4	0.029	0.029	0.0	0.0	0.857	0.0	0.0	0.0	0.0	0.086
5	0.0	0.0	0.0	0.07	0.047	0.721	0.07	0.0	0.07	0.023
6	0.064	0.0	0.0	0.0	0.043	0.0	0.894	0.0	0.0	0.0
7	0.0	0.045	0.0	0.0	0.068	0.0	0.0	0.705	0.0	0.182
8	0.053	0.0	0.053	0.0	0.053	0.079	0.026	0.0	0.658	0.079
9	0.0	0.0	0.0	0.024	0.049	0.0	0.0	0.098	0.0	0.829

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.053	0.026	0.763	0.026	0.079	0.0	0.0	0.0	0.026	0.026
3	0.029	0.029	0.0	0.912	0.029	0.0	0.0	0.0	0.0	0.0
4	0.0	0.029	0.0	0.0	0.886	0.0	0.0	0.0	0.0	0.086
5	0.023	0.023	0.0	0.07	0.023	0.767	0.047	0.0	0.023	0.023
6	0.064	0.0	0.0	0.0	0.021	0.0	0.915	0.0	0.0	0.0
7	0.0	0.045	0.0	0.0	0.023	0.0	0.0	0.727	0.0	0.205
8	0.053	0.0	0.026	0.026	0.0	0.079	0.0	0.0	0.763	0.053
9	0.0	0.0	0.0	0.024	0.049	0.024	0.0	0.098	0.0	0.805

Figure 85: Confusion matrices. To the left: BOE+No filtering (0.83), to the right: BOE+filtered with TAR(τ=0.08s, θ=2.4, R=1) (0.8525)

N-MNIST

With an appropriate choice of parameters we filtered 3,3% of events leading to an increase of 2,25% in accuracy score and 1,1% in the confidence of the prediction as shown in Figure 85.

The experiments were performed with only 1000 samples of the N-MNIST dataset used as training and 400 as testing set, this decision was due to computational reasons, but also to prove the impact when there is few data. When a lot of data is present it is easier for algorithms to generalise even in presence of noise.

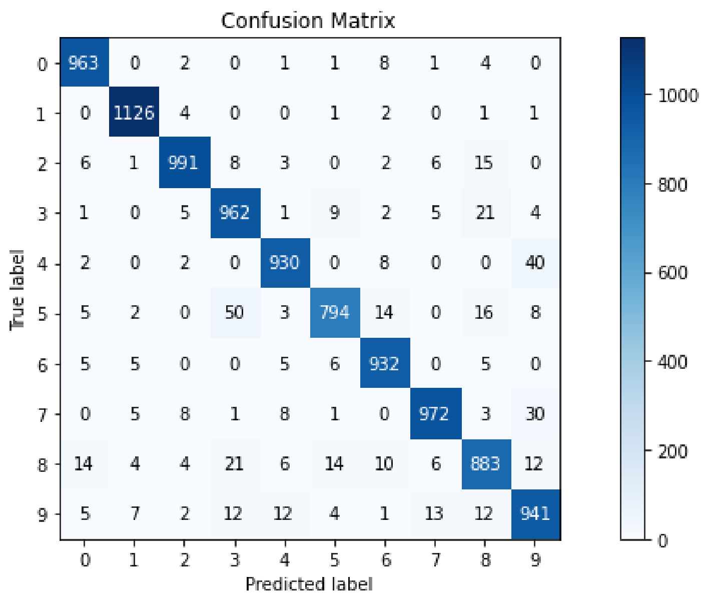
PokerDVS

Poker-DVS is a very small dataset developed by Serrano-Gotarredona and Linares-Barranco [63] including samples of the 4 pips of standard poker card set: diamond, club, hearts and spade. With this dataset we filtered 1% of events with TAR(τ=0.08s, θ=2, R=2) and achieved a 5,5% accuracy increase and 2% increase the confidence of the prediction. This higher increase in accuracy is also due to the very small size of the dataset.

BOE+No filtering achieved an accuracy score of 0.944 while BOE + TAR(τ=0.08s, θ=2, R=2) achieved an accuracy score of 1.0

B.3 CONSIDERATIONS

The method is really sensitive to the parameters, if we set the threshold θ too high we might end up filtering some events that may be valid. Decreasing the decay time constant decreases the "memory" of past events and therefore may also filter out valid events. Filtering out



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	963	0	2	0	1	1	8	1	4	0
1	0	1126	4	0	0	1	2	0	1	1
2	6	1	991	8	3	0	2	6	15	0
3	1	0	5	962	1	9	2	5	21	4
4	2	0	2	0	930	0	8	0	0	40
5	5	2	0	50	3	794	14	0	16	8
6	5	5	0	0	5	6	932	0	5	0
7	0	5	8	1	8	1	0	972	3	30
8	14	4	4	21	6	14	10	6	883	12
9	5	7	2	12	12	4	1	13	12	941

Figure 86: Confusion Matrix of the N-MNIST object classification TAR method. Accuracy is 0.9494%.

a high number of events with inappropriate parameters can be counterproductive and lead to worse results as events that are not really noise are filtered. To counteract also a "cold-start" we can allow the first k events regardless of the mask value. Usually a small number is enough $\simeq 10$ events.

B.4 A NEW METHOD

It is possible to use this filtering method as a replacement of the memory requirements in the HATS approach [88]. Indeed in this approach the purpose of memory cells is to reduce the influence of noise. This variant achieves a comparable performance while having no memory requirements. To prove this idea we performed some tests on the N-MNIST dataset. Results are in shown in Figure 86.

C NON-BIOINSPIRED PROCESSING APPROACHES

As it has been mentioned in previous chapters, many of the approaches in the event-based vision domain involve bio-inspired processing models like SNNs, since their asynchronicity makes them perfectly suited for event processing [8, 9]. In this thesis we only explored these bio-inspired processing approaches and compared the proposed algorithms with respect to other bio-inspired approaches and alternative approaches making use of statistical features or hierarchy of time surfaces, ignoring other approaches resorting to more traditional techniques like standard deep learning, random forests and so on. This is due to two main reasons: 1) in order to research further the underexploited bio-inspired domain (SNNs have been shown to be more computationally powerful than previous generations [5] but their full potential in practical applications is yet to be unlocked) and 2) due to the fact that neuromorphic hardware implementations of bio-inspired processing approaches promise to be able to operate faster and in a more energy efficient way than approaches resorting to traditional deep learning models.

Nonetheless, it is worth mentioning that at the current time this thesis was written, traditional deep learning methods far outperform bio-inspired approaches for many tasks including event-based vision-related tasks. This shortcoming can be attributed to the various reasons: the difficulty faced when training SNNs due to the non- differentiability at spike times which makes a transfer of the classical back-propagation mechanism impossible, the short amount of research done due to the novelty of such approaches, the short amount of "spiking datasets", among other reasons.

In this section we would like to briefly describe some of the existing approaches based on non-bioinspired processing like deep learning models, that have been adapted for event-based vision-related tasks and mention how bio-inspired approaches are getting inspiration from these kind of models to be able to better tackle these problems in the near future.

C.1 PROCESSING EVENTS

In deep learning, a common model type used to tackle visual-related tasks such as image classification and object detection is a Convolutional Neural Network (CNN). This model type achieves high performance metrics in a broad range of problems, but do so at a high compu-

Algorithm 15: Leaky surface frame integration

```
Initialise frame:
  $\forall x, y \in$ pixel grid,  $u(x, y) = 0$
Initialise time tracker:
  last_ts = $t_0$
foreach event $e_i = \langle x_i, y_i, p_i, t_i \rangle$ do
    Represent event:
      $u(x_i, y_i) += \Delta_{inc}$
    Compute delta timestamp:
      $\Delta_{ts} = t_i -$ last_ts
    Update time tracker:
      last_ts = $t_i$
    Decay values :
      $\forall x, y$,  $u(x, y) = f\,(u(x, y), \Delta_{ts})$
end
```

(a) Generic leaky surface frame integration algorithm. The Δ_{inc} is usually set to 1. The ensemble of pixel state values $u(x, y)$ at a given time t create a frame-like representation of the recent history of events, the relationship is established with a decay function f.

$$f = \max\{0, u(x, y) - \lambda \cdot \Delta_{ts}\}$$

(b) Leaky Surface with linear decay [89] [90]

$$f = u(x, y) \cdot \exp(-\Delta_{ts}/\tau)$$

(c) Leaky Surface with exponential decay, also called temporal context [62].

Figure 87: Common frame-integration methods of events used with deep learning approaches.

tational and energy cost. Due to the aforementioned problems in SNNs, some researchers have adapted this kind of networks for their use with event-based data by using frame-integration procedures to the events coming out of the event-camera. These approaches create a pseudo-frame which ignores temporal resolution [38], or create a pseudo-frame which includes temporal information encoded as some kind of decay. An example of the latter methods can be shown in Figure 87.

However, recent works have tried to ditch the handcrafted input representations like the aforementioned leaky surfaces, for a fully differentiable, end-to-end representation learning process that is learned end-to-end together with the downstream task, thus maximising performance. One of such approaches developed by Gehrig et al. [40], replaces a custom kernel function used to create the representation, with a Multi Layer Perceptron (MLP) with two hidden layers each with 30 units that learn input representations in a data-driven fashion and then uses an efficient look-up table at test time to speed up inference. A similar method, Matrix-LSTM [91], proposes a mechanism using

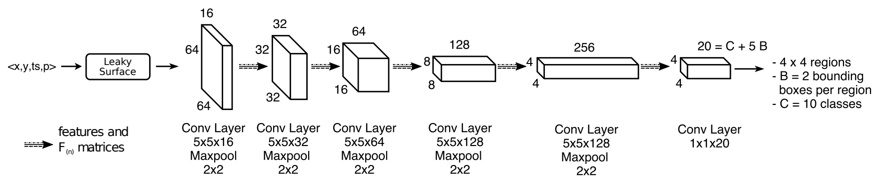

<x,y,ts,p>
Leaky Surface
16
64
64
32
32
32
64
16
16
128
8
8
256
4
4
20 = C + 5 B
4
4
- 4 x 4 regions
- B = 2 bounding boxes per region
- C = 10 classes
features and F(n) matrices
Conv Layer 5x5x16 Maxpool 2x2
Conv Layer 5x5x32 Maxpool 2x2
Conv Layer 5x5x64 Maxpool 2x2
Conv Layer 5x5x128 Maxpool 2x2
Conv Layer 5x5x128 Maxpool 2x2
Conv Layer 1x1x20

Figure 88: The fully-convolutional YOLE architecture. Image taken from [89].

a Long Short-Term Memory (LSTM) network as a convolutional filter over the 2D stream of events in order to accumulate pixel information through time and build 2D event representations specifically tailored for the task at hand.

C.2 APPROACHES FOR OBJECT DETECTION

Lately there has been a lot of research in traditional object detection algorithms, with almost every year having an algorithm presented at conferences or workshops that improves previous results in terms of speed of computation or performance metrics. Some of the most famous approaches are: Region Convolutional Neural Network (R-CNN) which despite its great accuracy is too slow to be used realtime due its copious forward passes required for a region proposal, You Only Look Once (YOLO) [68] which takes a regression approach to object detection and is thus capable of processing real-time videos with minimal delay while retaining respectable accuracy thanks to its single forward propagation prediction mechanism, Single Shot Multibox Detector (SSD), and most recently Recurrent-YOLO (ROLO) a single object tracking method that combines object detection and (LSTM) recurrent neural networks.

In 2019, Cannici et al. [89] proposed You Only Look Events (YOLE), an architecture relying on a standard CNN with the YOLO loss and a linear leak frame integration technique, and a fully-convolutional architecture based on the YOLE architecture (fc-YOLE) that smartly modifies the forward pass of fully convolutional architectures so that convolution and pooling operation are reformulated to avoid waste of power in computations by maintaining a state and recomputing only the features corresponding to regions affected by new incoming events while leaking allows past information to be forgotten. These changes are embedded in new event-based layer components: *e-conv* and *e-max-pool*. A general overview of the fc-YOLE architecture is shown in Figure 88.

C.3 APPROACHES FOR ROBOT CONTROL

Since the introduction of an autonomous land vehicle in a neural network (ALVINN) [92] in 1989, there has been a great interest in de-

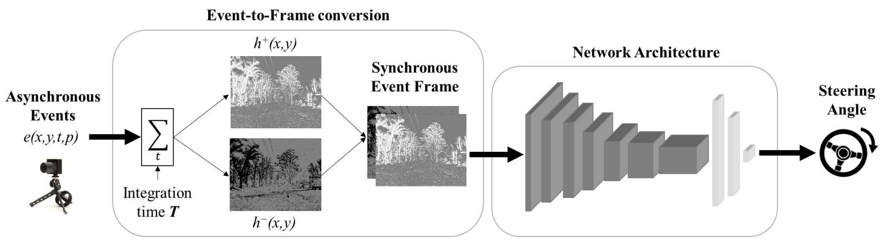


Figure 89: The steering-angle prediction architecture using a ResNet-inspired network. Image taken from [93].

veloping robust policies for autonomous driving and robot control. The event-based vision paradigm has fueled such interest thanks to its natural response to motion in the scene, high dynamic range, and low temporal resolution which brings a lot of potential advantages in the field. Examples of such approaches is the one proposed in 2016 by Moeys et al. [76], which used a CNN to steer the predator robot in the direction of the prey robot all while running on data from a DAVIS sensor, and the 2018 approach by Maqueda et al. [93] that presented a deep neural network approach which mapped event frames to a steering angle prediction for a vehicle by solving a regression task. An overview of the approach can be observed in Figure 89.

C.4 WHAT LIES AHEAD?

Besides the approaches in image classification and robot control, there have been multiple event-based vision applications relying on deep learning methods: image reconstruction [94], optical flow estimation [95], depth estimation [96] and many other applications, even a rock-paper-scissors playing agent [97], however, most of them require powerful GPUs to make tractable the lengthy and computationally expensive training procedures.

To overcome problem of scalable learning rules in SNNs, recent works focusing on converting pre-trained deep networks to SNNs [98] have achieved promising results even on complex tasks, while other works are still trying to find a way to efficiently implement spiking deep convolutional networks. Some of the successful approaches introduced include: direct training of SNNs using a backpropagation mechanism for deep SNNs that follows the same principles as in conventional deep networks, but works directly on spike signals and membrane potentials [99], a mapping of pre-trained ANNs parameters to SNNs, where the transfer function of ANNs is modified during training so that the network parameters can be mapped better to the SNN [100, 101], a novel method for adapting conventional Contrastive Divergence (CD) training algorithms for Deep Belief Networks (DBNs) with spiking neurons [102], just to cite a few.

Although these methods based on traditional ANNs have proved to produce state-of-the-art results, SNNs-only biologically inspired processing methods have developed their own identity and are increasingly abandoning a strong traditional ANN inspiration while at the same time increasing their performance. Recent work includes a fully deep SNN architecture combining STDP for the lowest layers and R-STDP for the highest layers [103], and a novel way of training deep SNNs by propagating a reward back through multiple layers of a dopamine-modulated SNN [104].

Some recent European projects, like the APROVIS3D project[1] involving 7 European universities, aim to develop new paradigms for biologically inspired vision, from sensing to processing, in order to help machines such as UAVs, autonomous vehicles, or robots gain high-level understanding from visual scenes without the use of traditional deep learning techniques, using instead neuromorphic implementations of SNNs.

1 https://www.chistera.eu/projects/aprovis3d

D DATASETS

D.1 N-MNIST

The Neuromorphic-MNIST (N-MNIST) is a spiking version of famous frame-based dataset MNIST which contains handwritten digits from 0 to 9. This dataset was obtained by Orchard et al. [34] though saccades in 3 directions, with a duration of 100ms each. Such process was made possible by mounting an event-based ATIS sensor on a motorized pan-tilt unit and having the sensor move while it views MNIST examples on an LCD monitor as shown in Figure 90.

The dataset contains the same 60000 train and 10000 test samples contained in the original dataset. An example is shown in Figure 91.

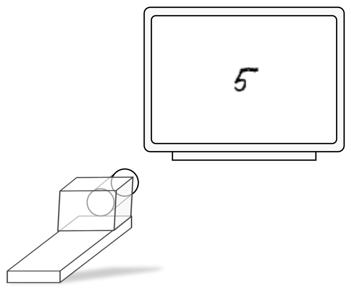


Figure 90: Data capture method for the N-MNIST dataset. The digit presentation lasts about 0.3 seconds and generates about $\simeq [2500, 6000]$ events.

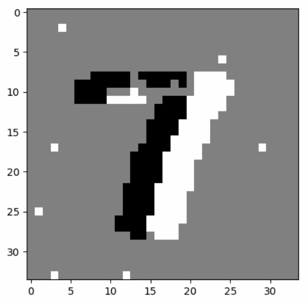


Figure 91: An example of a digit in the N-MNIST dataset. Given the fact that the dataset contains only events, the frame was reconstructed using a hard event segmentation, every 0.025s.

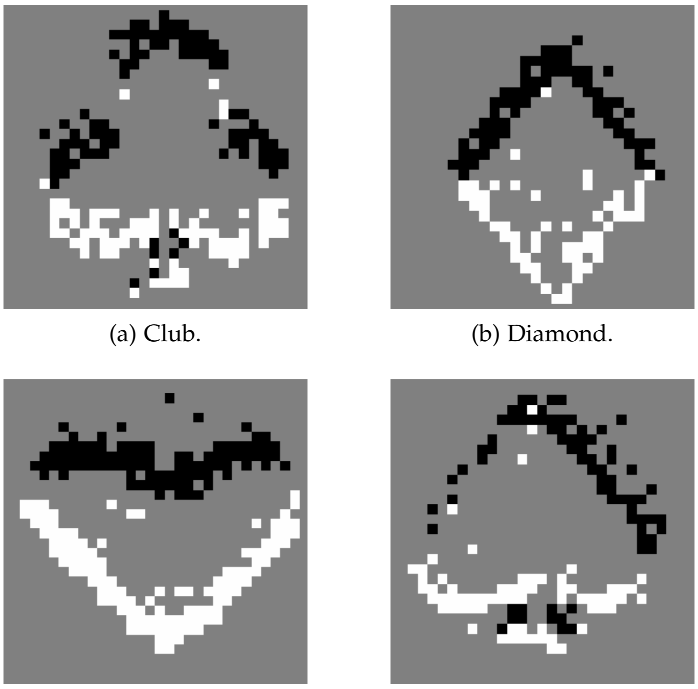

(a) Club.
(b) Diamond.
(c) Heart.
(d) Spade.

Figure 92: A sample for each of the 4 classes in the POKER-DVS. Each of the pseudo frames was reconstructed using a HES with a fixed time-slice of 0.02s.

D.2 POKER-DVS

The POKER-DVS [63], is a dataset developed by Teresa Serrano- Gotarredona and Bernabé Linares-Barranco at the Instituto de Microelectrónica de Sevilla. It was never intended to be used as a dataset useful for benchmarking, but was conceived just to illustrate a very high speed object recognition problem with event-driven convolutional neural networks.

The dataset was obtained by showing specially made poker card decks for 2–4 seconds in front of an improved DVS camera with faster event read-out scheme and better contrast sensitivity. Out of the single card presentations lasting about 20–30 ms, researchers tracked and isolated in an offline manner the poker pips of the cards (the markings that indicate the card's suit, i. e., spade, heart, diamond and club) to constitute the 131-samples of the dataset. An example for each group is shown in Figure 92.

The reason why the dataset contains poker cards is to honour Tobi Delbrück, one of the greatest researches in the field of event-based cameras and father of the DVS. He is a big fan of poker and usually organises a "Poker Night" at each year's Telluride Neuromorphic Engineering Workshop.

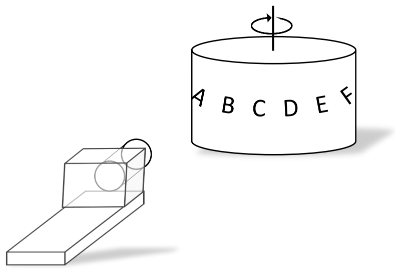
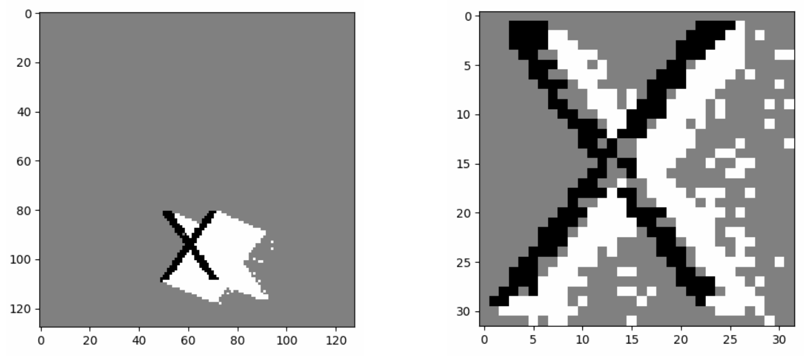


Figure 93: Data capture method for the DVS-BARREL dataset



(a) A moving character (b) A stabilized character

Figure 94: Samples of the DVS-BARREL dataset.

D.3 DVS-BARREL

The DVS-BARREL is a dataset developed by Orchard et al. [36] for testing the HFIRST object recognition architecture (Section 6.1.2). Its name derives from the fact that the characters contained in the dataset were obtained through a DVS sensor who saw characters moving on a motorised rotating barrel as shown in Figure 93.

The dataset is divided in two main groups which are shown in Figure 94:

- Moving
  It contains 76 recordings, where each character is tracked, and events not belonging to the character have been removed. Characters move in diagonal across the 128×128 pixel grid.

- Stabilized
  It contains the same 76 recordings, with the only difference that spikes are now relative to the center of the character, so the character appears to stay in a 32×32 pixel grid rather than moving across the screen.

BIBLIOGRAPHY

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] Jonathan W Mink, Robert J Blumenschine, and David B Adams. Ratio of central nervous system to body metabolism in vertebrates: its constancy and functional basis. *American Journal of Physiology-Regulatory, Integrative and Comparative Physiology*, 241 (3):R203–R212, 1981.

[3] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Third generation neural networks: Spiking neural networks. In *Advances in Computational Intelligence*, pages 167–178. Springer, 2009.

[4] Filip Ponulak and Andrzej Kasinski. Introduction to spiking neural networks: Information processing, learning and applications. *Acta neurobiologiae experimentalis*, 71(4):409–433, 2011.

[5] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[6] Ilias Sourikopoulos, Sara Hedayat, Christophe Loyez, François Danneville, Virginie Hoel, Eric Mercier, and Alain Cappy. A 4-fj/spike artificial neuron in 65 nm cmos technology. *Frontiers in neuroscience*, 11:123, 2017.

[7] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[8] Lea Steffen, Daniel Reichard, Jakob Weinland, Jacques Kaiser, Arne Roennau, and Rüdiger Dillmann. Neuromorphic stereo vision: A survey of bio-inspired sensors and algorithms. *Frontiers in neurorobotics*, 13:28, 2019.

[9] Veıs Oudjail and Jean Martinet. Bio-inspired event-based motion analysis with spiking neural networks. 2019.

[10] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[11] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[12] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[13] Kristina D Micheva, Brad Busse, Nicholas C Weiler, Nancy O'Rourke, and Stephen J Smith. Single-synapse analysis of a diverse synapse population: proteomic imaging methods and markers. *Neuron*, 68(4):639–653, 2010.

[14] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[15] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4): 500–544, 1952.

[16] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[17] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[18] Daniel A Butts, Chong Weng, Jianzhong Jin, Chun-I Yeh, Nicholas A Lesica, Jose-Manuel Alonso, and Garrett B Stanley. Temporal precision in the neural code and the timescales of natural vision. *Nature*, 449(7158):92–95, 2007.

[19] Simon Thorpe, Denis Fize, and Catherine Marlot. Speed of processing in the human visual system. *nature*, 381(6582):520–522, 1996.

[20] Qianhui Liu, Gang Pan, Haibo Ruan, Dong Xing, Qi Xu, and Huajin Tang. Unsupervised aer object recognition based on multiscale spatio-temporal features and spiking neurons. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[21] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[22] Sen Song, Kenneth D Miller, and Larry F Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3(9):919–926, 2000.

[23] Yuichiro Yada, Tatsuya Haga, Osamu Fukayama, Takayuki Hoshino, and Kunihiko Mabuchi. Multiplicative-stdp learning rule shows pathway specificity. *Transactions of Japanese Society for Medical and Biological Engineering*, 51(Supplement):R–169, 2013.

[24] Olivier Bichler, Manan Suri, Damien Querlioz, Dominique Vuillaume, Barbara DeSalvo, and Christian Gamrat. Visual pattern extraction using energy-efficient "2-pcm synapse" neuromorphic architecture. *IEEE Transactions on Electron Devices*, 59(8): 2206–2214, 2012.

[25] Sumit Bam Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, pages 1412–1421, 2018.

[26] Filip Ponulak and Andrzej Kasinski. Resume learning method for spiking neural networks dedicated to neuroprostheses control. In *Proceedings of EPFL LATSIS Symposium 2006, Dynamical Principles for Neuroscience and Intelligent Biomimetic Devices*, pages 119–120. Citeseer, 2006.

[27] Brandon Jennings. *Synchronization Analysis of Winner-Take-All Neuronal Networks*. PhD thesis, University of Pittsburgh, 2019.

[28] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

[29] Steve B Furber, David R Lester, Luis A Plana, Jim D Garside, Eustace Painkras, Steve Temple, and Andrew D Brown. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, 2012.

[30] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems*, 34(10):1537–1557, 2015.

[31] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38 (1):82–99, 2018.

[32] Alexander Neckar, Sam Fok, Ben V Benjamin, Terrence C Stewart, Nick N Oza, Aaron R Voelker, Chris Eliasmith, Rajit Manohar, and Kwabena Boahen. Braindrop: A mixed-signal

neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE*, 107(1):144–164, 2018.

[33] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128 $\times$ 128 120 db 15 μ s latency asynchronous temporal contrast vision sensor. *IEEE journal of solid-state circuits*, 43(2):566–576, 2008.

[34] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9:437, 2015.

[35] Christian Brandli, Raphael Berner, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. A 240$\times$ 180 130 db 3 µs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, 2014.

[36] Garrick Orchard, Cedric Meyer, Ralph Etienne-Cummings, Christoph Posch, Nitish Thakor, and Ryad Benosman. Hfirst: a temporal approach to object recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(10):2028–2040, 2015.

[37] Xi Peng, Bo Zhao, Rui Yan, Huajin Tang, and Zhang Yi. Bag of events: An efficient probability-based feature extraction method for aer image sensors. *IEEE transactions on neural networks and learning systems*, 28(4):791–803, 2016.

[38] Tobi Delbruck. Frame-free dynamic digital vision. In *Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, pages 21–26. Citeseer, 2008.

[39] Bo Zhao, Ruoxi Ding, Shoushun Chen, Bernabe Linares-Barranco, and Huajin Tang. Feedforward categorization on aer motion events using cortex-like features in a spiking neural network. *IEEE transactions on neural networks and learning systems*, 26(9):1963–1978, 2014.

[40] Daniel Gehrig, Antonio Loquercio, Konstantinos G Derpanis, and Davide Scaramuzza. End-to-end learning of representations for asynchronous event-based data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5633–5643, 2019.

[41] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[42] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2 (11):1019–1025, 1999.

[43] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE transactions on pattern analysis and machine intelligence*, 29(3):411–426, 2007.

[44] Judson P Jones and Larry A Palmer. An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex. *Journal of neurophysiology*, 58(6):1233–1258, 1987.

[45] Jan PH Van Santen and George Sperling. Elaborated reichardt detectors. *JOSA A*, 2(2):300–321, 1985.

[46] Garrick Orchard, Ryad Benosman, Ralph Etienne-Cummings, and Nitish V Thakor. A spiking neural network architecture for visual motion estimation. In *2013 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 298–301. IEEE, 2013.

[47] Runchun Mark Wang, Tara Julia Hamilton, Jonathan Tapson, and André van Schaik. A mixed-signal implementation of a polychronous spiking neural network with delay adaptation. *Frontiers in neuroscience*, 8:51, 2014.

[48] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in neural circuits*, 9:85, 2016.

[49] Elizabeth E Steinberg, Ronald Keiflin, Josiah R Boivin, Ilana B Witten, Karl Deisseroth, and Patricia H Janak. A causal link between prediction errors, dopamine neurons and learning. *Nature neuroscience*, 16(7):966, 2013.

[50] Eugene M Izhikevich. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral cortex*, 17(10):2443–2452, 2007.

[51] Michael A Farries and Adrienne L Fairhall. Reinforcement learning with modulated spike timing–dependent synaptic plasticity. *Journal of neurophysiology*, 98(6):3648–3665, 2007.

[52] Eleni Vasilaki, Nicolas Frémaux, Robert Urbanczik, Walter Senn, and Wulfram Gerstner. Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLoS computational biology*, 5(12), 2009.

[53] Claus Meschede, Zhenshan Bing, Florian Röhrbein, Kai Huang, and Alois C Knoll. A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in neurorobotics*, 12:35, 2018.

[54] Charles F Stevens. Quantal release of neurotransmitter and long-term potentiation. *Cell*, 72:55–63, 1993.

[55] H Sebastian Seung. Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, 40 (6):1063–1073, 2003.

[56] Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

[57] Riku Arakawa and Shintaro Shiba. Exploration of reinforcement learning for event camera using car-like robots, 2020.

[58] Earl K Miller, David J Freedman, and Jonathan D Wallis. The prefrontal cortex: categories, concepts and cognition. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 357(1424):1123–1136, 2002.

[59] Gerry Leisman. Brain networks, plasticity, and functional connectivities inform current directions in functional neurology and rehabilitation. *Funct Neurol Rehab Ergon*, 1(2):315–56, 2011.

[60] Robert Gütig and Haim Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature neuroscience*, 9 (3):420–428, 2006.

[61] Gerard Salton, Edward A Fox, and Harry Wu. Extended boolean information retrieval. *Communications of the ACM*, 26 (11):1022–1036, 1983.

[62] Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E Shi, and Ryad B Benosman. Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(7): 1346–1359, 2016.

[63] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. Poker-dvs and mnist-dvs. their history, how they were made, and other details. *Frontiers in neuroscience*, 9:481, 2015.

[64] Timothee Masquelier and Simon J Thorpe. Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS computational biology*, 3(2), 2007.

[65] Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. First-spike-based visual categorization using reward-modulated stdp. *IEEE transactions on neural networks and learning systems*, 29(12):6178–6190, 2018.

[66] Thomas Serre. Learning a dictionary of shape-components in visual cortex: comparison with neurons, humans and machines. 2006.

[67] Shoushun Chen, Polina Akselrod, Bo Zhao, Jose Antonio Perez Carrasco, Bernabe Linares-Barranco, and Eugenio Culurciello. Efficient feedforward categorization of objects and human postures with address-event image sensors. *IEEE transactions on pattern analysis and machine intelligence*, 34(2):302–314, 2011.

[68] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[69] Dario Floreano and Claudio Mattiussi. Evolution of spiking neural controllers for autonomous vision-based robots. In *International Symposium on Evolutionary Robotics*, pages 38–61. Springer, 2001.

[70] Dario Floreano, Jean-Christophe Zufferey, and Claudio Mattiussi. Evolving spiking neurons from wheels to wings. *Dynamic Systems Approach for Embodiment and Sociality*, 6(CONF): 65–70, 2003.

[71] Ezequiel Di Paolo. Spike-timing dependent plasticity for evolved robots. *Adaptive Behavior*, 10(3-4):243–263, 2002.

[72] Razvan V Florian. Biologically inspired neural networks for the control of embodied agents. *Center for Cognitive and Neural Studies (Cluj-Napoca, Romania), Technical Report Coneural-03-03*, 2003.

[73] Hani Hagras, Anthony Pounds-Cornish, Martin Colley, Victor Callaghan, and Graham Clarke. Evolving spiking neural network controllers for autonomous robots. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 5, pages 4620–4626. IEEE, 2004.

[74] Xiuqing Wang, Zeng-Guang Hou, Min Tan, Yongji Wang, and Liwei Hu. The wall-following controller for the mobile robot using spiking neurons. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 1, pages 194–199. IEEE, 2009.

[75] Fernando Perez-Peña, Arturo Morgado-Estevez, Alejandro Linares-Barranco, Angel Jimenez-Fernandez, Francisco Gomez-Rodriguez, Gabriel Jimenez-Moreno, and Juan Lopez-Coronado. Neuro-inspired spike-based motion: from dynamic vision sensor to robot motor open-loop control through spike-vite. *Sensors*, 13(11):15805–15832, 2013.

[76] Diederik Paul Moeys, Federico Corradi, Emmett Kerr, Philip Vance, Gautham Das, Daniel Neil, Dermot Kerr, and Tobi Del-

brück. Steering a predator robot using a mixed frame/event-driven convolutional neural network. In *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pages 1–8. IEEE, 2016.

[77] Hermann Blum, Alexander Dietmüller, Moritz Milde, Jörg Conradt, Giacomo Indiveri, and Yulia Sandamirskaya. A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor. *Robotics Science and Systems, RSS 2017*, 2017.

[78] Richard Evans. Reinforcement learning in a neurally controlled robot using dopamine modulated stdp. *arXiv preprint arXiv:1502.06096*, 2015.

[79] Bleema Rosenfeld, Osvaldo Simeone, and Bipin Rajendran. Learning first-to-spike policies for neuromorphic control using policy gradients. In *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2019.

[80] Jacques Kaiser, J Camilo Vasquez Tieck, Christian Hubschneider, Peter Wolf, Michael Weber, Michael Hoff, Alexander Friedrich, Konrad Wojtasik, Arne Roennau, Ralf Kohlhaas, et al. 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)*, pages 127–134. IEEE, 2016.

[81] Zhenshan Bing, Claus Meschede, Kai Huang, Guang Chen, Florian Rohrbein, Mahmoud Akl, and Alois Knoll. 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)*, pages 1–8. IEEE, 2018.

[82] J Camilo Vasquez Tieck, Pascal Becker, Jacques Kaiser, Igor Peric, Mahmoud Akl, Daniel Reichard, Arne Roennau, and Rüdiger Dillmann. Learning target reaching motions with a robotic arm using brain-inspired dopamine modulated stdp.

[83] Susanne Kunkel, Rajalekshmi Deepu, Hans Ekkehard Plesser, Bruno Golosio, Mikkel Elle Lepperød, Jochen Martin Eppler, Sepehr Mahmoudian, Jan Hahne, Dimitri Plotnikov, Claudia Bachmann, et al. Nest 2.12. 0. Technical report, Julich Supercomputing Center, 2017.

[84] Mitsuo Yokokawa, Fumiyoshi Shoji, Atsuya Uno, Motoyoshi Kurokawa, and Tadashi Watanabe. The k computer: Japanese next-generation supercomputer development project. In *IEEE/ACM international symposium on low power electronics and design*, pages 371–372. IEEE, 2011.

[85] Jacques Gautrais and Simon Thorpe. Rate coding versus temporal order coding: a theoretical approach. *Biosystems*, 48(1-3): 57–65, 1998.

[86] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[87] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[88] Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. Hats: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1731–1740, 2018.

[89] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Asynchronous convolutional networks for object detection in neuromorphic cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[90] Gregory Kevin Cohen. *Event-based feature detection, recognition and classification*. PhD thesis, Paris 6, 2016.

[91] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Matrix-lstm: a differentiable recurrent surface for asynchronous event-based data. *arXiv preprint arXiv:2001.03455*, 2020.

[92] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[93] Ana I Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5419–5427, 2018.

[94] Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. High speed and high dynamic range video with an event camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[95] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Ev-flownet: Self-supervised optical flow estimation for event-based cameras. *arXiv preprint arXiv:1802.06898*, 2018.

[96] Henri Rebecq, Guillermo Gallego, Elias Mueggler, and Davide Scaramuzza. Emvs: Event-based multi-view stereo—3d reconstruction with an event camera in real-time. *International Journal of Computer Vision*, 126(12):1394–1414, 2018.

[97] Iulia-Alexandra Lungu, Federico Corradi, and Tobi Delbrück. Live demonstration: Convolutional neural network driven by dynamic vision sensor playing roshambo. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–1. IEEE, 2017.

[98] Evangelos Stromatias, Miguel Soto, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data. *Frontiers in neuroscience*, 11:350, 2017.

[99] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.

[100] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. ieee, 2015.

[101] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*, 2015.

[102] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in neuroscience*, 7:178, 2013.

[103] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J Thorpe, and Timothée Masquelier. Combining stdp and reward-modulated stdp in deep convolutional spiking neural networks for digit recognition. *arXiv preprint arXiv:1804.00227*, 1, 2018.

[104] Zhenshan Bing, Zhuangyi Jiang, Long Cheng, Caixia Cai, Kai Huang, and Alois Knoll. End to end learning of a multi-layered snn based on r-stdp for a target tracking snake-like robot. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9645–9651. IEEE, 2019.

Bio-inspired Pattern Recognition and Detection from Event Cameras with Reward-Modulated Prototypes

Hidden for review
Hidden for review
Hidden for review
Hidden for review

Hidden for review
Hidden for review
Hidden for review
Hidden for review

Abstract

We propose a novel pattern recognition and detection approach combining three bio-inspired concepts: Event-Based Cameras (EBC) for sensing, and Spiking Neural Networks (SNN) and Reinforcement Learning (RL) for data processing, with an application to pattern recognition and detection. Event sensors provide a sparse representation of visual information. Based on this information, the approach takes the form of an HMAX*-like model, where the core of the learning is based on the Reward-modulated Synaptic Time Dependent Plasticity (R-STDP), that extends the Hebbian formulation of biological synaptic learning given by STDP. Our formulation of the learning rule extends R-STDP, which is a fundamental step towards allowing multi-class classification.*

We show that such an approach is able to successfully learn multiple class-specific prototypes given a labeled event-based training dataset. Such prototypes can successfully recognize and detect patterns from event data. To the best of our knowledge, this is the first proposal connecting the triple EBC-SNN-RL, and this work opens the way for further bio-inspired pattern analysis.

1. Introduction

Multiple disciplines have drawn inspiration from what can be observed in nature, result of centuries of optimization through natural selection. Neuromorphic engineering is a multi-disciplinary branch of engineering whose aim is to design systems and models whose architecture and design principles are based on those of biological nervous systems.

Event-Based Cameras (EBC) are neuromorphic devices that imitate the behaviour of the human retina. Instead of capturing images at a fixed frame-rate like conventional image sensors, each pixel inside the sensor operates independently and asynchronously, reporting relative changes in the light intensity under the form of events. Each event comprises the pixel coordinates (x, y) where the light intensity change occurred, a timestamp ts indicating the time when this change occurred, and a polarity p indicating whether the change in the light intensity was positive or negative. An event is thus formed by a tuple: (x, y, ts, p).

This kind of sensor benefits from a number of advantages, including a temporal resolution in the range of the microsecond, a very high dynamic range, no dazzling effect, and drastically reduced motion blur. Moreover, in non highly dynamic scenes, the visual information gathered is usually very sparse, which requires lower transmission bandwidth, storage capacity, processing time, and power consumption compared to conventional imaging sensors. Because of these outstanding features, EBC draw an increasing interest in the Computer Vision and Pattern Recognition communities, with applications including e.g. optical flow [23,35], motion estimation [14,29], deblurring [12,24] segmentation [19, 29], and feature detection [17]. Event-based sensors introduce a new paradigm which is not compatible with standard pattern recognition detection algorithms. Therefore, novel methods are required to process such unconventional data in order to unlock their potential.

In particular, Spiking Neural Networks (SNN), the third generation of neural networks, look particularly adapted to biologically-inspired event cameras since they also operate asynchronously. In SNN, information is encoded with rate or time and conveyed to other neurons by means of electrical pulses (spikes). This type of neural network has been shown to be computationally more powerful than previous generations of neural networks [16].

During training, SNNs generally do not rely on back-propagation and stochastic gradient descent, synapses are rather equipped with a *Spike-Timing-Dependent Plasticity* (STDP), which is a learning rule that implements a Hebbian learning [10], inspired from the organic hierarchical ventral path of the visual cortex. This rule updates synaptic weights according to causal links observed between presynaptic and postsynaptic spikes: a weight is increased (potentiated) when a postsynaptic spike occurs *shortly* after a presynaptic spike, and decreased otherwise. While this learning rule is generally implemented in unsupervised settings, it also can us used in supervised settings, where a postsynaptic spike is artificially triggered from selected output neurons by a supervisor in order to force synaptic potentiation. In this paper, however, we focus on yet another variation in a Reinforcement Learning [33] setting, inspired from the Reward-modulated STDP of Mozafari *et al.* [20].

This paper introduces an original approach for pattern recognition and detection that takes advantage of event cameras, spiking networks, and reward modulated STDP. Our contributions include the proposal of a first approach combining RL rule on SNN and EBC, in the form of a novel bio inspired pattern recognition and detection model inspired from HMAX, and a new formulation of the Reward-modulated Synaptic Time Dependent Plasticity (R-STDP) rule that enables multi-class classification. One important feature of this model is that because it can learn multiple prototypes per class, it handles will intra-class variations, which leaves flexibility in the definition of classes. To the best of our knowledge, this proposal is the first model that elegantly connects these three biologically-inspired concepts for pattern recognition, and it is the first bio-inspired approach of this kind for detection.

The remainder of the paper is organised as follows: Section 2 reviews work related to EBC, SNN, and RL, Section 3 introduces our proposed model to combine the three concepts in a unified model, Section 4 describes, comments, and discusses the experimental settings and results, and finally Section 5 summarizes our contributions and discusses future work.

2. Previous work

2.1. Event-based pattern classification

One of the first approaches for object classification using event data was introduced in 2011 by Chen *et al.* [3]. This approach is based on HMAX, a neurophysiologically plausible model of visual recognition initially proposed by Riesenhuber and Poggio [27] and later extended by Serre *et al.* [31], that has been a starting point for several work. The approach of Chen extracts size- and position-invariant line features with an HMAX model on 6 scales and 4 orientations, then with a modified line-segment Hausdorff-distance classifier, it computes the distance between the line segments of the test image and each one of the predefined library images to output a prediction.

In 2014, Zhao *et al.* [34] proposed an approach which extracts bio-inspired cortex-like features with a simple two-layered HMAX-inspired model and discriminates different patterns with the *tempotron* classifier of Gütig [9]. An important point to highlight here is that instead of processing data with a fixed number of events, this approach is event-driven, which means that there is no central clock, and therefore events are processed continuously, through an event-driven convolution with constant linear leakage.

A year later, in 2015, Orchard *et al.* [22] developed an approach based again on the HMAX model with Gabor filters at 12 different orientations called *HFirst*. Instead of relying on a non-linear pooling operation such as the `max` operation, this approach relies on the first spike received during computation. The number of spikes of each neuron in the HMAX C1 layer is counted for the input pseudo-image sample, and this count is translated into synaptic weights for the S2 layer to recognise a particular class.

After these two HMAX-inspired approaches, two other approaches have been developed in 2016: one is the Bag of Events (BOE) developed by Peng *et al.* [25], and the other is the Hierarchy Of Time-Surfaces (HOTS) proposed by Lagorce *et al.* [13].

In BOE, instead of lines, corners, or other visual features, the method is based on probability theory and is inspired on TF-IDF weighting scheme in information retrieval [28]. It combines the event frequency within a batch of events as a measure of *popularity* (TF) and the self-information as a measure of *specialty* (IDF) in a single metric that is then used for classification using an SVM. One of the advantages of this approach is that it implements an online learning algorithm, therefore it does not require the entirety of the training dataset to be provided in advance.

In HOTS, spatio-temporal features called *time surfaces* are extracted from the asynchronously-acquired dynamics of a visual scene, and then used to build a hierarchy of time surface prototypes at different time scales, whose activations are given as input for an object classifier.

More recently, in 2018, Sironi *et al.* [32] introduced HATS, another approach that introduces locally shared memory units retaining information of past events to create a descriptor made of an histogram of *averaged time surfaces*, which is then coupled with a simple linear SVM classifier for event-object classification.

In 2019, Cannici *et al.* [2] proposed You Only Look Events (YOLE), an architecture relying on a standard CNN using the YOLO loss [26] and a linear leak frame integration technique. This approach smartly modifies the forward pass of fully convolutional architectures so that convolution and pooling operations are reformulated to avoid waste of power in computations by maintaining a current state, and recomputing only the features corresponding to regions affected by new incoming events, while leaking allows past information to be forgotten. These changes are embedded in new event-based layer components: *e-conv* and *e-max-pool*.

2.2. SNN and RL for object classification

Introduced by Izhikevich in 2007 [11], the Dopamine-modulated STDP (DA-STDP) is inspired from the fact that neuromodulators such as dopamine are known to modify synaptic behaviour. The objective was to solve the distal problem, where the reward is not immediately received. Another rule, the Reward-modulated Synaptic Time Dependent Plasticity (R-STDP) proposed by Mozafari *et al.* [20] (which must not be confused with the DA-STDP rule proposed by Izhikevich, since DA-STDP is sometimes referred to as R-STDP, but they are different rules), was originally proposed to find patterns in static images with an RL-inspired rule that assigns weights a reward or punishment depending on whether the prediction is correct or not.

Finally, one of the latest approaches is the Multi-scale Spatio-Temporal (MuST) approach in 2020 by Liu *et al.* [15], that introduces a multi-scale approach to Gabor filters processing and learns patterns with the unsupervised learning rule STDP similarly to Masquelier *et al.* [18]. It should be noted that this approach, together with HFirst by Orchard *et al.* [22], and the approach developed by Zhao *et al.* [34], are some of the few approaches that combine event data with SNN. Going beyond Orchard and Zhao, this paper introduces an HMAX-inspired approach that innovatively integrates in the framework a novel multi-class RL-inspired learning rule: Reward-modulated Synaptic Time Dependent Plasticity (R-STDP), making our approach the first original model that successfully combines the three concepts of EBC, SNN, and RL.

3. Proposed approach

The model consists of an HMAX-like architecture designed for processing event data, and learn prototypes with R-STDP rule as illustrated Figure 1. A first part, prior to defining the model layers, consists in grouping the events with data-driven time slices in order to successively recreate a frame representation out of the events.

3.1. Event stream segmentation: preprocessing

Instead of using a fixed event count or a fixed time-slice as generally done with jAER tool by Delbrück [6], we use a soft event segmentation approach to dynamically segment events into groups. In particular, we use a threshold-based Motion Symbol Detector (MSD) [34].

As shown Figure 2, the MSD consists of a Leaky Integrate and Fire (LIF) neuron whose potential increases with the arrival of events, and decays exponentially in case of no input. The LIF neuron fires whenever a certain threshold is reached. The output spikes of this LIF neuron act as markers that separate batches of events. Once the events are partitioned by the MSD, events are grouped into time slices. This data-driven slice ignores the polarity and the temporal information of the events within the segment leaving as only information left the pixel coordinates of the events.

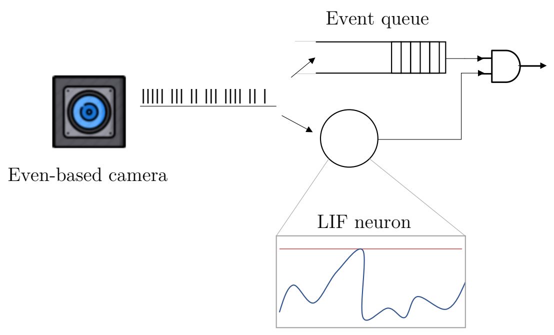


Figure 2: Motion Symbol Detector: when the threshold is reached, all the events accumulated in the queue are grouped in a batch and transferred to S1. The queue is then cleared and the neuron potential reset, so that the detector is ready to accumulate again.

3.2. Time slice filters – S1 and C1

The data-driven representation obtained in the previous step, is processed with 16 different Gabor filters – made of 4 scales (3, 5, 7, 9) and 4 orientations (0°, 45°, 90°, 135°) – located in the S1 layer that computes the strength of spatial features (see Figure 1).

After convolving the time slices with the filters with a same padding, the C1 layer downsamples the feature responses with a `max-pooling` operation over non-overlapping 2×2 regions of S1. This pooling is performed for each scale and orientation.

3.3. Encoding and scale fusion – from C1 to S2

The proposed approach includes an encoding strategy that transforms the pooled feature responses into spikes, inspired from Liu *et al.* [15]. We use a logarithmic Intensity-to-Latency (I2L) strategy, that maps spike times to values inversely proportional to the intensity of the response. The logarithmic version of I2L has proved a higher information entropy than the linear in Liu's work, which means that this version is more informative. Logarithmic I2L assigns a spike timestamp ts within a certain time window t_w based on the feature response value r with the following equation:

$$ts(r) = t_w \frac{\ln(r_{max}) - \ln(r)}{\ln(r_{max}) - \ln(r_{min})} \tag{1}$$

where r_{max} and r_{min} are respectively the maximum and minimum feature response value of C1. Once the responses are converted into spikes for each filter separately, multi-scale responses are merged – scales of origin of responses are discarded. Only the orientation and location are kept in S2.

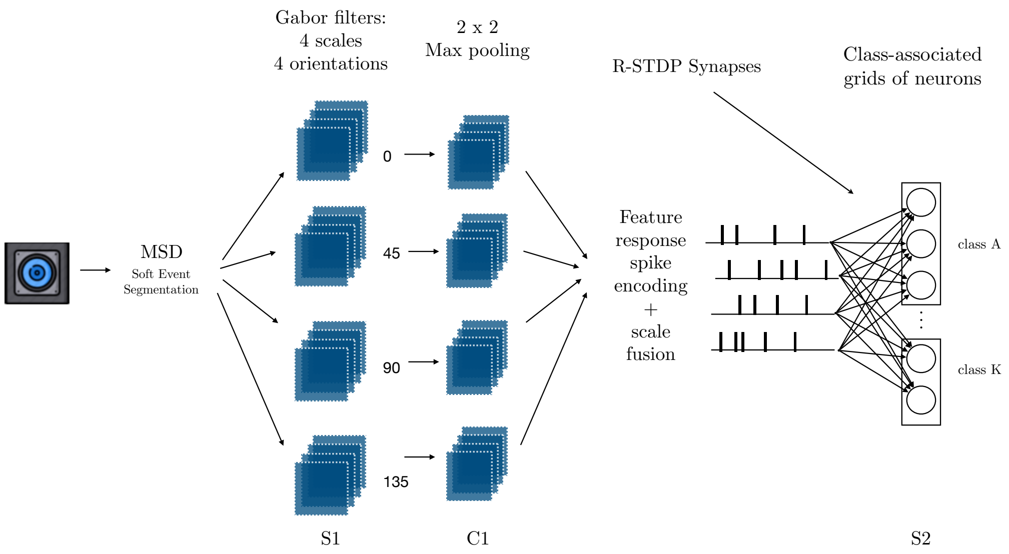


Figure 1: Overview of the proposed approach. The overall structure is inspired from the well-known HMAX model structure and used similarly as in HFirst [22]. The event segmentation idea is adapted from Zhao [34], while the multi-scale approach and encoding are suggested by MuST [15]. The RL-inspired learning algorithm (R-STDP) used to modify synaptic weights extends the proposal of Mozafari *et al.* [20]

.

3.4. Prototype-learning SNN (S2)

After spike encoding, spikes are processed with the S2 layer (see Figure 1), which consists of a convolutional spiking layer of Integrate-and-Fire (IF) neurons. All neurons in the convolutional spiking layer (grid) share the same weights. They have orientation-specific and location-specific weights, and they cover a 3×3 receptive field (input spiking area) in each orientation filter of C1, as illustrated Figure 3. Synaptic weights are updated with an extended version of the Reward-modulated Synaptic Time Dependent Plasticity (R-STDP) proposed by Mozafari *et al.* [20]. In Mozafari's work, wrong predictions can lead to a heavy decrease in the weights, which can ultimately result in *silent layers* (no output spikes at all). Since Mozafari's rule accepts silence as a valid prediction, and does not update weights in case of silence, this learning rule is subject to the *dead neuron* problem. We believe that this is the reason why the original paper only tackled two-class classification problems: in our multi-class classification scenario test, the network ultimately reached a point in which it was silent for most classes, which is obviously a problem. To address this problem, we introduce a variation of the existing learning rule [20, eq. (6,7)], which does not only update the grid of the neuron that spiked, but it also may increase weights in all the grids associated to the correct class.

In case of a reward (correct prediction), the grid weights

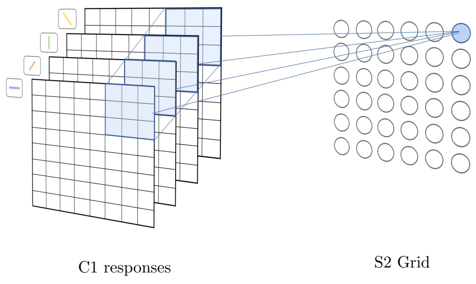


Figure 3: The S2 grid contains neurons that cover the entire C1 spiking regions. In the particular case shown in the illustration, the grid weights are represented as a 3x3x4 matrix which covers a 3×3 region of 4 different orientations.

update is given by the following equation:

$$\Delta w_{ij} = \begin{cases} A_r^- \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) \geq t_i \\ A_r^+ \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) < t_i \end{cases} \tag{2}$$

while in case of punishment (wrong prediction), the weight updates are instead the following:

$$\Delta w_{ij} = \begin{cases} A_p^- \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) \leq t_i \\ A_p^+ \cdot w_{ij} \cdot (1 - w_{ij}) & \text{if } t^f(j) > t_i \end{cases} \tag{3}$$

in addition, if there is a wrong prediction or silence,

$$w_{ij} + \epsilon, \qquad \forall w_{ij} \in Grids(label) \tag{4}$$

where j is the winning neuron with spike time $t^f(j)$, t_i is the eventual time of arrival of a spike at the neuron's receptive field region i, and w_{ij} is the synaptic weight between the input region and the neuron, and $Grids(label)$ indicates the ensemble of Grids pre-allocated to the given label.

With these update rules, if a reward is received (Eq. 2), only the grid containing the neuron that spiked undergoes a weight modification. This update rule increases the weights by a factor of $A_r^+ > 0$ if at a certain position in the neuron's receptive field, there was a $C1$ response spike before the S2 neuron spiked (successful situation: true positive). Given that the prediction was correct (since a reward is received), this increase in the weights aims to reinforce the association of certain features' position and orientation to the class. In parallel, this same update rule updates weights by a factor of $A_r^- < 0$, if at a certain position, there was not a $C1$ response spike before the S2 neuron spiked (silence: true negative). This decrease in the weights is triggered so that S2 becomes sensitive only to relevant positions and orientations, and ignores the rest of the spikes given that they might not be class-relevant.

When instead a punishment is received (Eq. 3), the weights are updated by a factor of $A_p^- < 0$ if at a certain position, there was a $C1$ response spike before the winner neuron spiked (false positive). This decrease prevents grids from being sensitive to irrelevant stimuli, not useful to distinguish the different classes as it has just lead to a wrong prediction. If at a certain position, $C1$ has emitted some spikes after the winner neuron spiked (false negative), then the weights are increased $A_p^+ > 0$ so that the neuron becomes sensitive to the a different pattern, which might be correct for the associated class. In addition, in case of punishment or silence (Eq. 4), all weights in grids associated to the correct class are increased by a small amount ϵ so that class grids are more sensitive to this stimulus for the next predictions.

This formulation enables the method to address multi-class classification problems. In order to achieve a distinction between the different classes, multiple grids of neurons (K grids per class with $K \geq 1$) are needed. When $K > 1$, the model is able to learn different prototypes per class as shown in Figure 4.

3.5. Prediction

The first neuron in any grid that reaches the threshold and emits a spike is considered the winning neuron in a Winner-Take-All (WTA) strategy. Since each grid is associated to a class and we have multiple grids, each one belonging to a certain class, the prediction in the proposed approach is naturally the class associated to the grid containing the winning spiking neuron or silence in case of no spike:

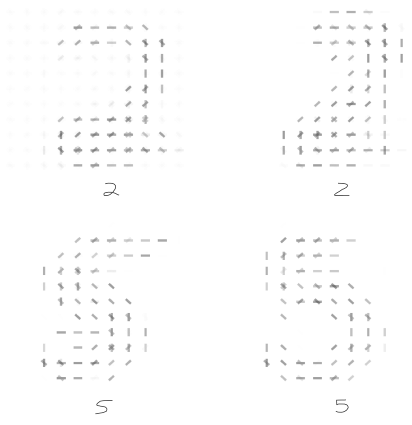


Figure 4: Example of two different prototypes of two digits learnt during the training mechanism. The approach is able to learn prototypes that represent the class, but also is able to learn different representations of the same class as it can be observed from the illustrated representations. Top: the 2 on the left includes a loop while the second is more straight and Z-shaped. Bottom: the 5 on the left is more S-shaped and the 5 on the right is more rounded on the middle-lower part.

$$\begin{aligned} &\hat{y} = label(Grid_m) \Leftrightarrow \exists j \forall l,\, t^f(j) \leq t^f(l) \wedge j \in Grid_m \\ &\quad\vee \\ &\hat{y} = \emptyset \Leftrightarrow \nexists j, t^f(j) \end{aligned} \tag{5}$$

where m is the index of a grid $\in \{0, K-1\}$, j and l indicate S2 neuron indices and $t^f(\cdot)$ indicates the firing time of a neuron.

4. Experimental Results

In order to test the efficacy of the proposed method, we have tested it with the N-MNIST [21], POKER-DVS [30], and POSTURE-DVS dataset [3]. The first dataset contains digits moving in a 34×34 pixel array obtained through saccades, the next features pips of cards moving on the screen while the latter features human postures on a 128×128 pixel array.

4.1. Parameter selection

The number of grids is chosen according to how diverse the classes are (i.e. the intra-class dissimilarity). For instance, if we want to classify digits from 0-9 in two classes: even and odd, we would need at least 5 grids per class since elements within a class do not share visual similarities. In the standard 10-class case, since each number corresponds to a different class, we initially selected 1 grid per class and later increased the number of grids per class to give the model opportunity to recognise variations of the same

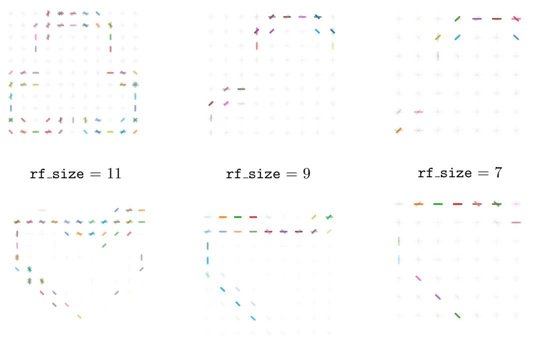


Figure 5: Prototype comparison with different receptive field sizes on the POKER-DVS dataset. The learning rule was able to find the class-specific patterns shown above for receptive fields 11×11, 9×9, and 7×7 while it was unable to find one with a 5×5 receptive field.

class. Therefore in our experiments, we tried $k = 2$ grids per class for the datasets.

The receptive field was selected for all datasets to be the smallest region which can contain the entire target pattern. By selecting a too small receptive field, the algorithm is likely to fail at identifying a pattern that is able to properly separate the classes. For instance, a 5×5 receptive field for N-MNIST might identify a semicircle that looks like an horizontally flipped C. Since this pattern occurs in numbers: zero, two, three and five, it will not be helpful for differentiating these classes.

In Figure 5, it is possible to observe experiments on the size of the receptive field.

4.2. Class-specific prototypes

The experiments revealed interesting results, since the method was able to learn the proper weights that represent a whole class. Figure 6 shows the weights learnt during training for the various datasets. In order to make the weights more interpretable, a line in the orientation of the associated Gabor filter is plotted with an opacity proportional to the weight value.

4.3. Going beyond classification: pattern detection

Pattern detection is a harder problem than classification since the goal is both classification and location identification (by means of a bounding box) of a single, or multiple patterns in a visual scene. By selecting S1 to be the same size as the event sensor, an S2 receptive field big enough to contain the entire pattern to classify, and by keeping track of the first neuron inside the grid to spike, the proposed approach can be thought as a parallelised sliding-window object detection approach with a fixed square size (which equals to the size of the receptive field of the neurons).

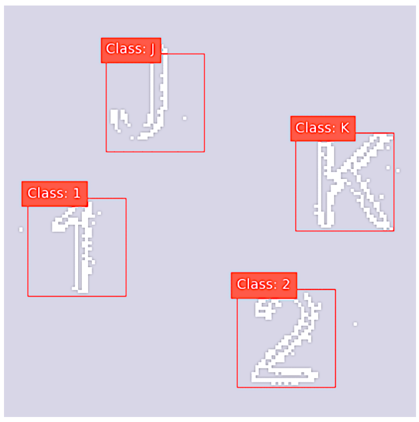


Figure 7: Multi-class detection on the original DVS-BARREL recording.

This is due to the fact that S2 neurons are capable of identifying a class pattern along all the input region thanks to their weight sharing strategy. Note that the proposed algorithm, as it has been explained to this point, is only capable of identifying one pattern in the scene and not multiple patterns. This limitation is easily overcome by allowing multiple temporal winners during the prediction phase and inhibit neurons to fire in regions where an object has already been identified.

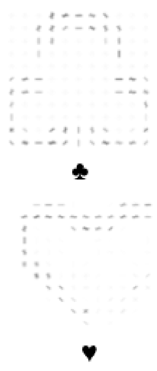
The importance of setting a good neuron firing threshold, becomes important in this context as the algorithm is more prone to false alarms with partial activations of the pattern. Figure 7 shows detection and classification results on the DVS-BARREL dataset [22].

4.4. Performance

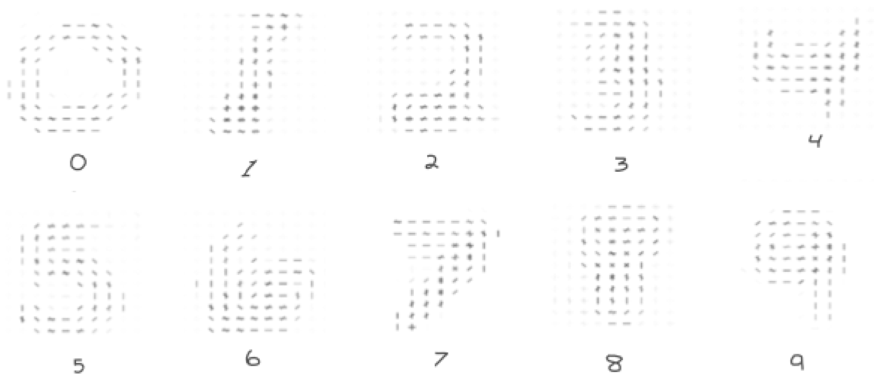
A comparison of the performance of the proposed approach and other bio-inspired and simple methods can be found in Table 1. The table indicates that the proposed approach works particularly well compared to some other bio-inspired approaches, and performs particularly well in datasets where the elements belonging to a class are easily generalised by a handful of prototypes like the POKER-DVS dataset, while it does not achieve expected performance on the N-MNIST due to the difficulty of generalising the handwritten digits with a small number of prototypes, this could be improved by considering more prototypes per class.

4.5. A Reinforcement Learning approach or not?

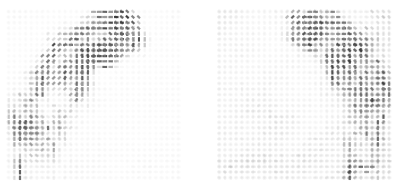
It can be questioned whether this approach actually belongs or not to Reinforcement Learning. The fact that the reward is constructed based on the training samples is a major argument against this categorization and may lead to believe is just a way of supervision. However, this approach is not only able to learn with training samples – in case there are no class labels, a positive reward can be assigned, mak-



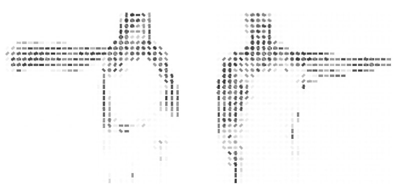
(a) POKER-DVS



(b) N-MNIST



(c) POSTURE-DVS: bending



(d) POSTURE-DVS: hand1

Figure 6: Learnt prototypes with the proposed approach.

Table 1: Classification performance of different approaches (best performances in bold).

Approach	POKER-DVS	N-MNIST	Useful for
Zhao *et al.* [34]	93.00%	86.60%	Classification
BOE [25]	93.00%	70.43%	Classification
HFIRST [22]	94.00%	71.15%	Classification
HOTS [13]	97.50%	80.8%	Classification
MUST [15]	99.00%	**89.7**%	Classification
This work	**99.09** %	84.4%	Classification & Detection

ing the method behave just like the standard unsupervised STDP learning rule. Class labels can then be assigned to the each one of the learnt patterns. Once class labels are assigned, the model starts to learn more appropriate patterns which help better differentiate the classes.

Because the method is reward-based and learning-based through trial-and-error actions (where the prediction is the action taken), the best definition for the approach is that of RL: an agent learns by trial and error using feedback from its own actions and experiences to maximise a notion of reward. In fact, the prototypes are created in a trial-and-error way by reinforcing weights, which leads to correct predictions.

5. Conclusion and future work

This paper introduces a novel approach for pattern recognition that combines event-based data, spiking neural networks, and reinforcement learning with a new formulation of the Reward-modulated Synaptic Time Dependent Plasticity. To the best of our knowledge, this proposal is the first one that connects the three biologically-inspired concepts of EBC-SNN-RL for a pattern recognition task. Moreover, it is the first approach of this kind that performs pattern detection.

Event-based camera are not yet widely available. However, by using libraries such as Gehrig's [8] to convert standard video to events streams, our approach can be applied to standard video datasets, although without fully taking advantage of event sensors.

The frame integration technique used in the approach ignores the temporal resolution within a batch of events, this was chosen to avoid more complexity in the software simulations, however it is evident this is not ideal, especially in the event-based paradigm, for this reason, it should be worth experimenting with leaky surfaces that better consider the temporal resolution of events by introducing a leakage function f depending on the internal state u of each of the pixels and the Δ_{ts} between the timestamps. Examples of leaky surfaces are the linear decaying surface Eq. 6 [2, 4] or the exponentially decaying surface Eq. 7 [13].

$$f = \max\{0, u(x, y) - \lambda \cdot \Delta_{ts}\} \tag{6}$$

$$f = u(x, y) \cdot exp(-\Delta_{ts}/\tau) \tag{7}$$

Although it is out of the scope of this paper, it is worth highlighting that eventually, such an approach is intended to be deployed of specific hardware such as IBM's TrueNorth [1], Intel's Loihi [5], SpiNNaker [7], or specific neuromorphic hardware.

References

[1] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE transactions on computer-aided design of integrated circuits and systems*, 34(10):1537–1557, 2015. 8

[2] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Asynchronous convolutional networks for object detection in neuromorphic cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1656–1665, 2019. 2, 8

[3] Shoushun Chen, Polina Akselrod, Bo Zhao, Jose Antonio Perez Carrasco, Bernabe Linares-Barranco, and Eugenio Culurciello. Efficient feedforward categorization of objects and human postures with address-event image sensors. *IEEE transactions on pattern analysis and machine intelligence*, 34(2):302–314, 2011. 2, 5

[4] Gregory Kevin Cohen. *Event-based feature detection, recognition and classification*. PhD thesis, Paris 6, 2016. 8

[5] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018. 8

[6] Tobi Delbrück. Frame-free dynamic digital vision. In *Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, pages 21–26, 2008. 3

[7] Steve B Furber, David R Lester, Luis A Plana, Jim D Garside, Eustace Painkras, Steve Temple, and Andrew D Brown. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, 2012. 8

[8] Daniel Gehrig, Mathias Gehrig, Javier Hidalgo-Carrió, and Davide Scaramuzza. Video to events: Recycling video datasets for event cameras. In *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, June 2020. 7

[9] Robert Gütig and Haim Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature neuroscience*, 9(3):420–428, 2006. 2

[10] D. O. Hebb. The organization of behaviour, a neuropsychological theory. *The American Journal of Psychology*, 63(4):633–642, 1950. 1

[11] Eugene M Izhikevich. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral cortex*, 17(10):2443–2452, 2007. 3

[12] Zhe Jiang, Yu Zhang, Dongqing Zou, Jimmy Ren, Jiancheng Lv, and Yebin Liu. Learning event-based motion deblurring. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1

[13] Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E Shi, and Ryad B Benosman. Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(7):1346–1359, 2016. 2, 7, 8

[14] Daqi Liu, Alvaro Parra, and Tat-Jun Chin. Globally optimal contrast maximisation for event-based motion estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1

[15] Qianhui Liu, Gang Pan, Haibo Ruan, Dong Xing, Qi Xu, and Huajin Tang. Unsupervised aer object recognition based on multiscale spatio-temporal features and spiking neurons. *IEEE Transactions on Neural Networks and Learning Systems*, 2020. 3, 4, 7

[16] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997. 1

[17] Jacques Manderscheid, Amos Sironi, Nicolas Bourdis, Davide Migliore, and Vincent Lepetit. Speed invariant time surface for learning to detect corner points with event-based cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[18] Timothee Masquelier and Simon J Thorpe. Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS computational biology*, 3(2), 2007. 3

[19] Anton Mitrokhin, Zhiyuan Hua, Cornelia Fermuller, and Yiannis Aloimonos. Learning visual motion segmentation using event surfaces. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1

[20] Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. First-spike-based visual categorization using reward-modulated stdp. *IEEE transactions on neural networks and learning systems*, 29(12):6178–6190, 2018. 2, 3, 4

[21] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9:437, 2015. 5

[22] Garrick Orchard, Cedric Meyer, Ralph Etienne-Cummings, Christoph Posch, Nitish Thakor, and Ryad Benosman. Hfirst: a temporal approach to object recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(10):2028–2040, 2015. 2, 3, 4, 6, 7

[23] Liyuan Pan, Miaomiao Liu, and Richard Hartley. Single image optical flow estimation with an event camera. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1

[24] Liyuan Pan, Cedric Scheerlinck, Xin Yu, Richard Hartley, Miaomiao Liu, and Yuchao Dai. Bringing a blurry frame alive at high frame-rate with an event camera. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[25] Xi Peng, Bo Zhao, Rui Yan, Huajin Tang, and Zhang Yi. Bag of events: An efficient probability-based feature extraction method for aer image sensors. *IEEE transactions on neural networks and learning systems*, 28(4):791–803, 2016. 2, 7

[26] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2

[27] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999. 2

[28] Gerard Salton, Edward A Fox, and Harry Wu. Extended boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983. 2

[29] Yusuke Sekikawa, Kosuke Hara, and Hideo Saito. Eventnet: Asynchronous recursive event processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[30] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. Poker-dvs and mnist-dvs. their history, how they were made, and other details. *Frontiers in neuroscience*, 9:481, 2015. 5

[31] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE transactions on pattern analysis and machine intelligence*, 29(3):411–426, 2007. 2

[32] Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. Hats: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1731–1740, 2018. 2

[33] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, An Introduction*. 2018. 2

[34] Bo Zhao, Ruoxi Ding, Shoushun Chen, Bernabe Linares-Barranco, and Huajin Tang. Feedforward categorization on aer motion events using cortex-like features in a spiking neural network. *IEEE transactions on neural networks and learning systems*, 26(9):1963–1978, 2014. 2, 3, 4, 7

[35] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

Biologically-inspired robot control

Rafael Mosca[1] and Jean Martinet[1]

Abstract— **We introduce a novel biologically-inspired approach for robot control. The approach is based on Spiking Neural Networks (SNN), a model of asynchronous artificial neural networks that is closer to biology than formal networks traditionally used in machine learning. The sensing is done with *event-based visual sensors* (silicon retinas), that mimic biological retinas by outputting asynchronous events for each pixel based on individual luminance changes. We illustrate this approach on aerial robot navigation scenarios. The contribution is three-fold: (1) our SNN are trained with a Reinforcement-Learning-based version of the standard Spike-Timing Dependant Plasticity (STDP) widely used to optimise weights in spiking networks, that is the *dopamine-modulated STDP*, (2) the sensing is done with event-based cameras, and (3) to the best of our knowledge, this is the first approach combining the promising triple: Event-Based Cameras, Spiking Neural Networks, and Reinforcement Learning. Experimental simulation results of drone navigation for object/line following scenarios demonstrate superior results of the proposed approach compared to other Reinforcement Learning approaches.**

I. INTRODUCTION

Event-Based Cameras (EBC), also called silicon retinas or dynamic vision sensors, are biologically inspired vision sensors that do not collect frames at a fixed frame rate, instead, these kind of sensors output changes on pixel-level brightness. By doing so, these kind of cameras offer some considerable advantages over standard frame-based cameras, namely a very high dynamic range, absence of motion blur, no redundancy in visual information, low power consumption, and a latency in the microsecond range.

SNN are a type of artificial neural networks that is closer to the biological model of neurons as information is conveyed to other neurons by means of electrical pulses called *spikes*. In this type of neural network, the information is processed asynchronously, and it has been shown to be more computationally powerful than previous generations of neural networks [11].

EBC emit events asynchronously, which makes SNN well suited for their processing [15]. We believe that EBC coupled with SNN make a sound and promising combination for autonomous robots or vehicles that need energy efficient and fast real-time calculations. However, because EBC and SNN are fundamentally different from standard frame-based cameras and formal neuron models widely used in deep learning, previously developed algorithms cannot be directly applied. Hence, there is a need for new algorithms developed *ad-hoc* for these models.

In this paper, we introduce an original biologically-inspired approach for robot control, based on Event-Based Cameras and Spiking Neural Networks trained with Reinforcement Learning, which has been largely unexplored and unexploited in this context. Therefore this work is intrinsically interdisciplinary since it brings together and connects contributions in robot control, machine learning, computational neurosciences, and computer vision.

The remainder of this paper is organised as follows. Section II reviews work related to the use of EBC, SNN, and RL for robot control. Sections III and IV describe our proposed approach and the experimental settings and results. Section V provides conclusions and future directions.

II. RELATED WORK

Applications of SNN for robot control started in the early 2000's, with a number of attempts for autonomous navigation applications. Floreano and Mattiussi used evolutionary methods for vision-based SNN controllers that performed navigation tasks in different contexts: a small two-wheeled robot navigating in a rectangular arena with textured walls [5], and blimps and micro-flyers navigating in an indoor scenario [6]. Di Paolo [3] performed a first attempt in 2002 to control a robot coupling STDP with an evolutionary strategy, and Florian [7] showed in 2003 that networks using STDP evolved faster (in terms of generations) than networks with static synapses. Hagras et al. [8] used in 2004 an adaptive crossover and mutation genetic algorithm on a robot with nine ultrasound sensors and four bump sensors, to converge faster to a solution that exhibited the desired edge-following behaviour. In 2009, Wang et al. [21] built a robot that is capable of following a wall by using an SNN controller and 16 evenly distributed ultrasonic sensors, even when obstacles are present.

As a step beyond these successful experiments on SNN-based robot control, the use of event-based vision sensors offer considerable advantages over standard frame-based cameras. In 2013, Perez-Peña et al. [13] used EBC data to reproduce intended movements performed by humans with a neuroinspired algorithm (SVITE: Spike-based VITE). Some years later, in 2016, Moeys et al. [12] used a CNN with data from an EBC to control a "predator" and follow another which acted as a "prey", while in 2017, Blum et al. [2] implemented a spike-based robotic controller on neuromorphic hardware that is able to perform reactive obstacle avoidance and target acquisition in an unknown environment using as sensory input just an EBC and an Inertial Measurement Unit.

This work has been supported by the French government, through the EUR DS4H Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-17-EURE-0004.

[1]Both authors are with the Université Côte d'Azur, CNRS, I3S `rafael.mosca.aguilar@gmail.com` and `jean.martinet@univ-cotedazur.fr`

Lately, there has been an increased interest in RL with SNN [19], and it seems natural to apply this paradigm to SNN for robot control. Biological synapses release a type of chemical messengers called *neurotransmitters* upon the arrival of an action potential from the presynaptic neuron. These neurotransmitters are diffused across the small space between the two neurons which is called *synaptic cleft* and then bind to receptors on the dendrites of the post-synaptic neurons. As a result of this binding, the postsynaptic neuron is influenced in its behaviour and its spike-generating activity is either excited or inhibited. It has been shown also that the release of a particular neurotransmitter called *dopamine*, is causally linked to the expected future reward [16]. Indeed, unexpected rewards activates midbrain dopamine neurons, and the magnitude of the dopamine release depends on unexpectedness of the reward. The "credit assignment problem" is a central problem in the reinforcement learning literature. This problem consists in knowing how and which of the cues and actions received prior to the reward should be credited for it, even in situations where the reward may be delayed. [9] showed how modulating the *LTP* and *LTD* components of *STDP* with the neuromodulator is a reasonable solution to this problem. The modulation of STDP with the neuromodulator proposed by [9] in 2007, transforms the standard unsupervised learning paradigm of STDP, into a reward-based learning paradigm, hence the name *R-STDP*.

Vasilaki et al. [18] used in 2009 an RL-based SNN to control a robot in the Morris water navigation task, the famous task used to study spatial learning and memory, which consists in finding an invisible or visible platform that allows the agent to escape the water. Evans [4] used in 2015 R-STDP on a robot with range and touch sensors that had to learn the correct behaviour to collect food items (avoid poison, empty containers, etc.), while other researchers like Rosenfeld et al. [14] attempted in 2019 neuromorphic control using policy gradient-based algorithms for SNN.

Finally, Kaiser et al. [10] developed in 2016 a framework to evaluate neural self-driving vehicle applications, which is a milestone for emergence of approaches combining EBC and SNN. This framework, shown in Fig. 1, consists in using a ROS as a communication middleware between a world simulator like Gazebo or V-REP, which represents both the environment and the agent, and a ROS that controls the agent. In such framework, the simulation data is passed to the SNN who then takes the actions needed (such as steering commands), and sends them to the simulated vehicle.

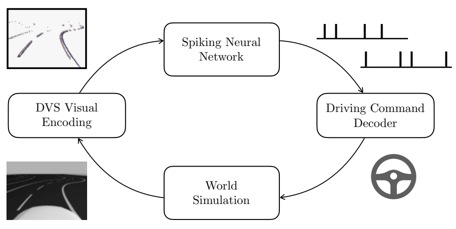


Fig. 1: The general simulation framework for robot control.

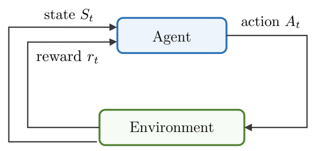
This framework inspired Bing et al. [1] in 2018 to improve the original paper baseline. Instead of picking a set with handpicked-weights, the authors translated policies learned with standard Deep Q-Learning to an SNN and explored SNN learning rules such as R-STDP. With the same inspiration, Bing et al. also explored R-STDP on a snake-like robot used to perform target tracking tasks. Such experiments inspired in turn Tieck et al. [17] in 2019 to use R-STDP on a robotic arm that needs to reach a target and to perform manipulation tasks. We also use this simulation framework to demonstrate our proposed approach with vision-based object/line following tasks.

III. PROPOSED APPROACH

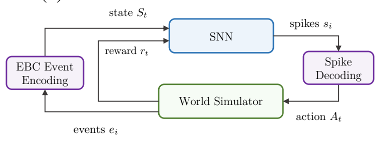
We describe in this section our biologically-inspired approach for robot control, based on EBC and SNN trained with R-STDP. In order to demonstrate the usefulness of R-STDP in robot control, and its superiority with respect to traditional RL algorithms, we investigate two scenarios for an aerial robot (quadrotor) equipped with an EBC, with the constraint of using no other visual source than the EBC for the algorithms.

A. Overview

From a high-level overview, the simulations are based on the self-driving vehicle simulation framework (see Fig. 1). Our simulation framework is pictured Fig. 2-(b), adapted from a standard RL framework shown Fig. 2-(a). We investigated a ball-following scenario, and a line-following scenario.



(a) The standard RL framework.



(b) The high level simulation framework for an SNN agent.

Fig. 2: RL Frameworks for standard and SNN agents.

In the ball-following scenario (see 3), the objective is to train the robot controller to keep a moving ball always in the camera's field of view, similarly to a *follow-me* drone that tracks a target-object (generally a person in outdoor sports settings) despite erratic and high-speed movements. In our experiments, we chose the target object to be a ball, which during the simulations follows a pre-defined path (a Bezier curve defined by 8 control points which are positioned in a four-branch star shape) that is unknown to the robot.

In the line-following scenario (see 4), the objective is to train the robot controller to follow a line painted on the

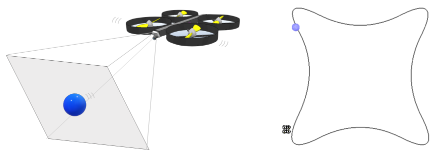


Fig. 3: Ball-following scenario: the ball moves at a speed of 0.075 m/s along a path that is invisible to the drone. The altitude of the drone is fixed at 2.4m from the ground. The episode finished whenever the ball is not in the field of view or when a lap to the path has been completed.

floor. The robot moves forward at a constant speed and the controller must learn how and when to rotate left or right in order to precisely follow the line, and to keep it centered with a 90° orientation.

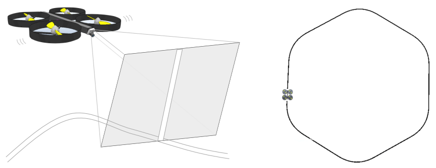


Fig. 4: Line-following scenario: the drone moves forward at a speed of 0.075 m/s. The altitude of the drone is fixed at 2.4m from the ground. The reason why the line is shaped like an hexagon is due to the fact that the curves created by such shape have an angle greater than 90 degrees which allows the estimation of orientation of the line in the scene. This is needed to assign a reward. In scenarios with different shapes, the drone must be close enough to the line so as to only see one line and not two like in a very arched curve. The episode finishes when the line is not in the drone's field of view or a lap has been completed.

B. *The standard* RL *agents*

For comparison purposes, we used Deep RL agents as standard reinforcement learning agents, that take a vector as input, and output an integer (in case of discrete-action compatible approaches), or a floating point vector (in case of continuous-action compatible approaches).

In order to create the input vector, we did not create an image out of the events as it would make the input dimension too large and possibly too complex to learn a policy. Instead, the EBC pixel grid is tessellated, so that each region measures a binary pixel activity during a simulation step (time pooling), resulting in a binary grid.

C. *The* SNN *agent*

Our SNN agent needs to transform received events into a particular event representation. Here again, the EBC pixel grid is tessellated, however each region measures the number of spikes during a simulation step. Each regions is associated to a spiking neuron, that encodes the incoming event count using a *rate coding* (see below) mechanism and generate an output spike train that depends on the input. These spiking neurons are fully connected (all-to-all) to the output layer. The synapses connecting both layers of spiking neurons have synaptic weights that are trained with the R-STDP/dopamine-modulated learning rule.

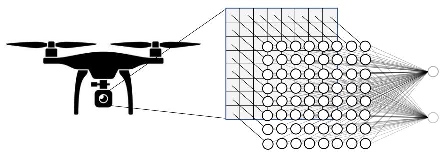


Fig. 5: SNN agent network overview.

D. *Encoding and decoding*

Our proposed approach includes (1) a way of representing information of the environment (be it physical or simulated) in the activity of a neuron (*neural encoding*), and (2) a mechanism that interprets neuron activity and translates that activity into electrical signals that drive actuators (*neural decoding*). Rate coding (or *frequency coding*) is the most widely used coding scheme that assumes that most, if not all information about the stimulus, is contained in the firing rate of the neuron. A popular coding mechanism is the spike-count rate. This approach, also referred to as temporal average, is obtained by counting the number of spikes during a certain time interval and by dividing this number with the duration T of the time interval:

$$\lambda = \frac{n_{spikes}}{\Delta T} \tag{1}$$

One advantage of rate coding is that it can be described with a Poisson model:

$$\Pr\{\text{ k spikes in interval } T\} = \frac{\lambda^k}{k!}e^{-\lambda} \tag{2}$$

Regarding decoding, we use a muscle modelling decoding. In case of a continuous action set, the action model contains a pair of neurons in the output layer for each degree of freedom. The robot is controlled by the SNN agent based on the output spikes. The action to be taken is based on the amount of spikes emitted from each neuron:

$$\frac{n_{spikes}(positive) - n_{spikes}(negative)}{n_{max\_spikes}} \tag{3}$$

where positive and negative indicate the positive and negative options for each degree of freedom, and the max number of spikes depends on the simulation time and the length of refractory periods of neurons.

E. *Dopmine-modulated STDP*

The modulation of STDP with dopamine proposed by [9] is made possible by using 2 variables to describe each synapse: the *synaptic weight/strength* s, and c, an enzyme important for plasticity that acts as an ”synaptic eligibility trace”, in addition to a global variable d that is used to describe the extracellular dopamine.

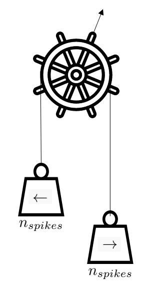


Fig. 6: SNN control action model, based on spike count difference.

$$\frac{\partial}{\partial t}c(t) = -\frac{c(t)}{\tau_c} + STDP(\tau)\,\delta(t - t_{pre/post} \tag{4}$$

$$\frac{\partial}{\partial t}d(t) = -\frac{d(t)}{\tau_d} + DA(t) \tag{5}$$

$$\frac{\partial}{\partial t}s(t) = c(t)\,d(t) \tag{6}$$

Eq. 4 represents the eligibility trace, a concept borrowed from the SARSA(λ) approach in classical RL theory. This trace decays with a time constant τ_c and suffers modifications according to the STDP rule and when the Dirac delta $\delta(t)$ is enabled at pre and post firing times $t_{pre/post}$. STDP(τ) indicates a function which returns a value depending on the post-pre firing delta also called inter-spike interval $\tau = t_{post} - t_{pre}$. Eq. 5 controls the extracellular dopamine, that decays with a time constant τ_d and increases with a term DA that acts upon the arrival of the reward. Finally, the synaptic strength in Eq. 6 is modified only when dopamine is present ($d > 0$), and when it is eligible for the reward ($c > 0$).

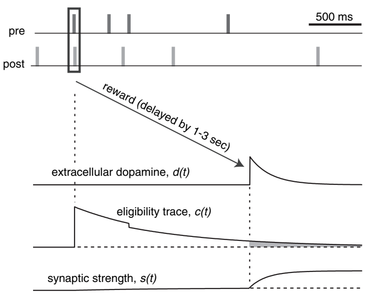


Fig. 7: Dopamine-modulated STDP [9].

IV. EXPERIMENTAL RESULTS

As explained in previous sections, with the environment abstraction, all that is needed is an implementation of the agent that encapsulates the logic for taking actions in the environment. The agents implemented for the ball-following and the line-following task are the following:

- a Deep Q-Network (DQN),
- an SNN agent using R-STDP synapses for learning
- a Deep Deterministic Policy Gradient agent (DDPG).

The implementations have been realised using CoppeliaSim 4.0.0, NEST 2.20.0 for SNN simulation, ROS Melodic, with Python 2.7, all under Ubuntu 18.04 LTS.

A. BALL-FOLLOWING *scenario*

In order to demonstrate the difference in performance between the several available methods, we compute the average euclidean error/distance to the target. Since we want the ball centered, it should be positioned close to the middle of the pixel grid. To better illustrate the performance of the algorithms, we plot the density heatmap of the ball positions, with a darker colour to indicate positions that have been visited more often during simulations, and inversely. The absence of colour (paper colour/white), indicates that such positions were not visited at all. On the sides of the density plot, the marginal distributions are displayed. These distributions describe the projected position of the ball from the point of view of a single axis.

In Fig. 8, we compare the density heatmaps for the algorithms that converged to a solution. DDPG is not displayed as this algorithm did not converge to a solution after over 24h of training. In Figs. 9 and 10, we show how the error and convergence varies with the training episodes for SNN-R-STDP and respectively, DQN. The approach using an SNN with static weights is not shown since it is not trained and converges from the very beginning; DDPG is not shown either since it did not converge.

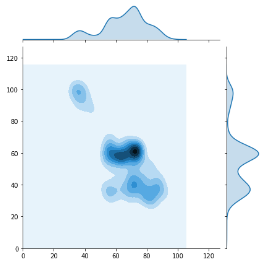
Episodes with steps above the red bar are considered completed. The number of steps in a completed simulation are not fixed, they might slightly vary from run to run. This is due to the simulated nature of the experiments.

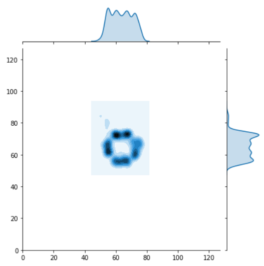
Notice that the chosen metric of euclidean distance is more informative than the average position of the ball. As it can be observed from the density of the simulation with fixed weights, the average position obtained by the approach would be still very close to the center, but this is only due to the fact that positive and negative errors will compensate each other. By using the euclidean distance with the formula below, such errors do not compensate each other because the error is always positive.

$$\text{Error} = \frac{1}{T}\sum^{T}\sqrt{(x_i - x_t)^2 + (y_i - y_t)^2} \tag{7}$$

As it can be observed from the density plots, even though DQN has the peak of its distribution on the desired position, it does does not always succeed on keeping the ball at the center, the visited positions cover almost all the pixel grid and the wide distribution indicates a lower precision. This density plot represents the result achieved after 14h of training.

If we observe the density plot with the fixed weights, we see a kind of loop around the center, which results in a wobbly and wide plateau on the marginal distribution. On the other hand, the SNN approach with R-STDP achieves a nearly gaussian distribution which is peaked at exactly the desired position and does not visit other nearby states as much as in the case with static weight or DQN.


(a) DQN. Corresponding to an average euclidean error to the target of 21.48.


(b) SNN + fixed weights. Corresponding to an average euclidean error to the target of 10.02

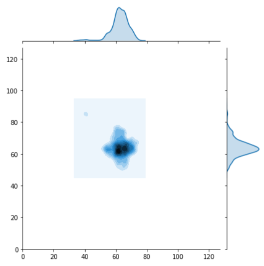
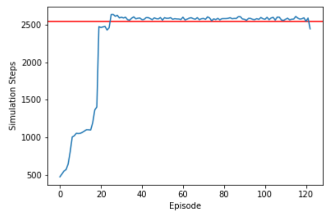

(c) SNN + R-STDP. Corresponding to an average euclidean error to the target of 6.77.

Fig. 8: In figures (a)-(c), it is possible to appreciate the density function of the ball positions during an episode. It is worth noticing that SNN + R-STDP outperforms the other methods.


(a) After about 30 episodes ($\simeq$ 60000 simulation steps), the algorithm converged to a solution which finished the episode at every time.

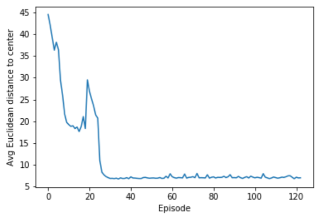
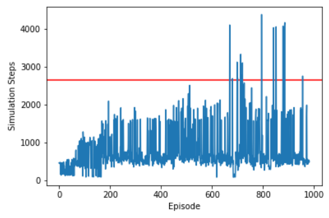

(b) Graph of the average euclidean distance to the center on an episode.

Fig. 9: Results for the SNN-R-STDP agent.


(a) After about 650 episodes ($\simeq$ 696000 simulation steps), the algorithm successfully concluded an episode.

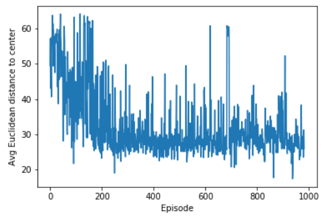

(b) Graph of the average euclidean distance to the center on an episode.

Fig. 10: Results for the DQN agent.

In all the simulations, we can see that on all the density plots, the upper left side contains some visited states, due to the initialisation: the ball is not positioned at the center when the episode begins, causing these out-of-center positions.

B. Line-following *scenario*

In the line-following scenario, we aim to keep the line centered and with an angle of $90°$ (vertical). In this scenario, we plot the mean position of the center of the line on one axis and the orientation on the other one, the objective would be to have the density concentrated on the middle point of the density plot which is $(63, 90)$. The results are shown in Fig. 11 and Table II.

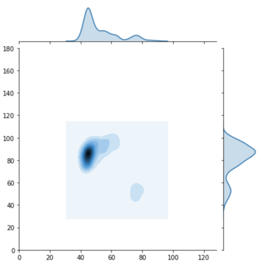
Also in this scenario, DDPG failed to converge while SNN-R-STDP achieved the best result.

V. Conclusions

In this paper, we have described a novel biologically-inspired approach for robot control, with an application to aerial robot navigation. The approach uses Event-Based Cameras for sensing, and Spiking Neural Networks trained with Reinforcement Learning for processing the visual information and controlling the robot. To the best of our knowledge, this is the first approach combining the promising triple EBC, SNN, and RL. We believe that this bio-inspired triple is important and promising, and yet only little attention has be given to such a combination in the communities,

Approach	Avg Pos.	Avg. Error	Steps to conv.
Static	(61.61, 63.66)	10.02	0
SNN+static init	(60.74, 63.69)	8.55	0
SNN	(61.20, 62.62)	6.77	60000
DQN	(62.88, 65.17)	21.48	696000

TABLE I: Results for the ball-following scenarios. Note that since static weights do not need training and already encode a human solution for the problem, the drone converges from the very beginning. In this table we include an extra case in which the SNN uses the static weights as initialisation, while this performs better than just using static weights, the initialisation heavily conditions the solution and underperforms w.r.t. a solution found with equal weights initialisation.


(a) Density plot for a DQN approach.

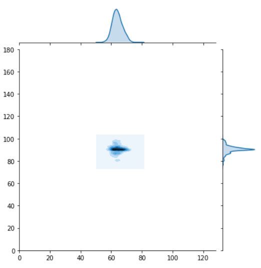

(b) Density plot for an SNN-R-STDP approach.

Fig. 11: Density plots for the line-following scenarios.

possibly because of its interdisciplinary nature.

In our simulations, we experienced convergence issues, and long simulation times. It should be emphasised that in the long term, SNN models are to be implemented and executed on dedicated neuromorphic hardware (e.g. IBM TrueNorth, Intel Loihi), and therefore enabling full advantages of SNN.

As future work, we are interested in adapting the *Spike Timing Dependent Delay Plasticity (STDDP)* learning rule of Wang et al. [20] to our RL framework, so as to take advantage of an R-STdDP learning rule, which instead of modifying synaptic weights, modifies synaptic delays in a RL-inspired manner.

Approach	Avg Pos.	Steps to conv.
SNN	(63.04, 88.47)	10
DQN	(54.28, 84.87)	648

TABLE II: Results for the line-following scenarios.

REFERENCES

[1] Z. Bing, C. Meschede, K. Huang, G. Chen, F. Rohrbein, M. Akl, and A. Knoll. 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)*, pages 1–8. IEEE, 2018.

[2] 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. *Robotics Science and Systems, RSS 2017*, 2017.

[3] E. Di Paolo. Spike-timing dependent plasticity for evolved robots. *Adaptive Behavior*, 10(3-4):243–263, 2002.

[4] R. Evans. Reinforcement learning in a neurally controlled robot using dopamine modulated stdp. *arXiv preprint arXiv:1502.06096*, 2015.

[5] D. Floreano and C. Mattiussi. Evolution of spiking neural controllers for autonomous vision-based robots. In *International Symposium on Evolutionary Robotics*, pages 38–61. Springer, 2001.

[6] D. Floreano, J.-C. Zufferey, and C. Mattiussi. Evolving spiking neurons from wheels to wings. *Dynamic Systems Approach for Embodiment and Sociality*, 6(CONF):65–70, 2003.

[7] R. V. Florian. Biologically inspired neural networks for the control of embodied agents. *Center for Cognitive and Neural Studies (Cluj-Napoca, Romania), Technical Report Coneural-03-03*, 2003.

[8] H. Hagras, A. Pounds-Cornish, M. Colley, V. Callaghan, and G. Clarke. Evolving spiking neural network controllers for autonomous robots. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 5, pages 4620–4626. IEEE, 2004.

[9] E. M. Izhikevich. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral cortex*, 17(10):2443–2452, 2007.

[10] J. Kaiser, J. C. V. Tieck, C. Hubschneider, P. Wolf, M. Weber, M. Hoff, A. Friedrich, K. Wojtasik, A. Roennau, R. Kohlhaas, et al. 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)*, pages 127–134. IEEE, 2016.

[11] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[12] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbrück. Steering a predator robot using a mixed frame/event-driven convolutional neural network. In *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pages 1–8. IEEE, 2016.

[13] F. Perez-Peña, A. Morgado-Estevez, A. Linares-Barranco, A. Jimenez-Fernandez, F. Gomez-Rodriguez, G. Jimenez-Moreno, and J. Lopez-Coronado. Neuro-inspired spike-based motion: from dynamic vision sensor to robot motor open-loop control through spike-vite. *Sensors*, 13(11):15805–15832, 2013.

[14] B. Rosenfeld, O. Simeone, and B. Rajendran. Learning first-to-spike policies for neuromorphic control using policy gradients. In *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2019.

[15] L. Steffen, D. Reichard, J. Weinland, J. Kaiser, A. Roennau, and R. Dillmann. Neuromorphic stereo vision: A survey of bio-inspired sensors and algorithms. *Frontiers in neurorobotics*, 13:28, 2019.

[16] E. E. Steinberg, R. Keiflin, J. R. Boivin, I. B. Witten, K. Deisseroth, and P. H. Janak. A causal link between prediction errors, dopamine neurons and learning. *Nature neuroscience*, 16(7):966, 2013.

[17] J. C. V. Tieck, P. Becker, J. Kaiser, I. Peric, M. Akl, D. Reichard, A. Roennau, and R. Dillmann. Learning target reaching motions with a robotic arm using brain-inspired dopamine modulated stdp. In *EEE 18th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC)*, pages 54–61, 2019.

[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*, 5(12), 2009.

[19] A. Vigneron and J. Martinet. A critical survey of stdp in spiking neural networks for pattern recognition. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2020.

[20] R. M. Wang, T. J. Hamilton, J. Tapson, and A. van Schaik. A mixed-signal implementation of a polychronous spiking neural network with delay adaptation. *Frontiers in neuroscience*, 8:51, 2014.

[21] X. Wang, Z.-G. Hou, M. Tan, Y. Wang, and L. Hu. The wall-following controller for the mobile robot using spiking neurons. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 1, pages 194–199. IEEE, 2009.