Computational models of object motion detectors accelerated using FPGA technology

Department of Computer Science
School of Science and Technology
Nottingham Trent University

Pedro Miguel Baptista Machado

A thesis submitted in partial fulfilment of the requirements of
The Nottingham Trent University for the degree of Doctor of Philosophy

July, 2021

I would like to dedicate this PhD thesis to my wife Laura, children Elisa and Ana, my parents, sister and my grandmother Helena Machado. Laura has been by my side, giving me support and encouragement during this long PhD journey. I am truly thankful for having you in my life, and please remember that this PhD degree is also yours. Elisa, I tried my best to be there and play with you while you were growing up. Ana, thank you for all the support you gave me and for stepping up and helping mom looking after Elisa when I was working on my PhD.

Obrigado mãe pelo apoio e força que sempre me deste. À memória da minha avó que sempre me apoiou no meu percurso académico.

Acknowledgements

I want to thank my supervisory team and independent assessor; Professor Martin McGinnity, for his immeasurable wisdom, being my mentor, extended support, encouragement and invaluable guidance during the PhD journey; Dr Andreas Oikonomou for being there to support me when I needed the most and for the long, very productive brainstorms and guidance; Professor Eiman Kanjo for her support and guidance, Dr João Filipe Ferreira who was my 2^{nd} Supervisor; Professor Ahmad Lotfi for his support, encouragement and guidance. I also want to thank my PhD thesis' non-official reviewers and friends;

Dr Robert Ranson (mentor and friend), Dr John Wade (long-time friend), Dr Farhad Fassihi Tash, Dr (to be) Francisco Lemos (lifetime friend), Dr Lorenzo Ferrara, Dr Isibor Kennedy Ihianle and Samuel Brandenburg. Their feedback and suggestions were crucial to finalise the PhD thesis.

A big thanks to my colleagues and friends; *Dr David Adama, Dr Kayode Owa, Dr Salisu Yahaya, Dr Kofi Appiah, Dr Nikesh Lama, Dr Peter FitzGerald, Dr Neil Sculthorpe, Dr Filipe Neves dos Santos, Dr Micael Couceiro, Dr David Portugal, Dr Raquel Santos (close friend), Dr Alexey Petrushin, Jorge Rosário (close friend), Luís Costa (lifetime friend and mentor), Nuno Semedo (lifetime friend), João Reis*

(lifetime friend), Hugo Faria (lifetime friend) and Bruno Santos (lifetime friend) and many other names that were not enumerated here. Each one of them have contributed in their way to help me to reach this important milestone.

Abstract

The detection of moving objects is a trivial task when performed by vertebrate retinas, yet a complex computer vision task. This PhD research programme has made three key contributions, namely: 1) a multi-hierarchical spiking neural network (MHSNN) architecture for detecting horizontal and vertical movements, 2) a Hybrid Sensitive Motion Detector (HSMD) algorithm for detecting object motion and 3) the Neuromorphic Hybrid Sensitive Motion Detector (NeuroHSMD) , a real-time neuromorphic implementation of the HSMD algorithm.

The MHSNN is a customised 4 layers Spiking Neural Network (SNN) architecture designed to reflect the basic connectivity, similar to canonical behaviours found in the majority of vertebrate retinas (including human retinas). The architecture, was trained using images from a custom dataset generated in laboratory settings. Simulation results revealed that each cell model is sensitive to vertical and horizontal movements, with a detection error of 6.75% contrasted against the teaching signals (expected output signals) used to train the MHSNN. The experimental evaluation of the methodology shows that the MHSNN was not scalable because of the overall number of neurons and synapses which lead to the development of the HSMD.

The HSMD algorithm enhanced an existing Dynamic Background subtraction (DBS) algorithm using a customised 3-layer SNN. The

customised 3-layer SNN was used to stabilise the foreground information of moving objects in the scene, which improves the object motion detection. The algorithm was compared against existing background subtraction approaches, available on the Open Computer Vision (OpenCV) library, specifically on the 2012 Change Detection (CDnet2012) and the 2014 Change Detection (CDnet2014) benchmark datasets. The accuracy results show that the HSMD was ranked overall first and performed better than all the other benchmarked algorithms on four of the categories, across all eight test metrics. Furthermore, the HSMD is the first to use an SNN to enhance the existing dynamic background subtraction algorithm without a substantial degradation of the frame rate, being capable of processing images 720×480 at 13.82 Frames Per Second (fps) (CDnet2014) and 720×480 at 13.92 fps (CDnet2012) on a High Performance computer (96 cores and 756 GB of RAM). Although the HSMD analysis shows good Percentage of Correct Classifications (PCC) on the CDnet2012 and CDnet2014, it was identified that the 3-layer customised SNN was the bottleneck, in terms of speed, and could be improved using dedicated hardware.

The NeuroHSMD is thus an adaptation of the HSMD algorithm whereby the SNN component has been fully implemented on dedicated hardware [Terasic DE10-pro Field-Programmable Gate Array (FPGA) board]. Open Computer Language (OpenCL) was used to simplify the FPGA design flow and allow the code portability to other devices such as FPGA and Graphical Processing Unit (GPU). The NeuroHSMD was also tested against the CDnet2012 and CDnet2014 datasets with an acceleration of 82% over the HSMD algorithm, being capable of

processing 720 × 480 images at 28.06 fps (CDnet2012) and 28.71 fps (CDnet2014).

Contents

List of Figures

List of Tables

List of Acronyms

ADAS Advanced-Driver Assist Systems. 1, 5, 29, 35, 57, 62, 187

AER Address Event Representation. 65–67, 69

AI Artificial Intelligence. 44, 63, 75, 146, 148, 160

ALM adaptive logic module. xvii, 150–152, 171, 172

ALU adaptive look-up-table. 150, 171–173

ANN Artificial Neural Network. 3, 5, 64

ASIC Application-Specific Integrated Circuits. 4

BRIEF Binary Robust Independent Elementary Features. 53

BS Background subtraction. xii, xv, xix, 2, 3, 5–9, 29, 35–37, 39–41, 43, 73, 74, 76, 79, 98, 112–120, 124, 127, 129, 130, 142, 164, 180, 183–187

CAPOA Content-Adaptive Progressive Occlusion analysis. 54

CCR Correct Classifications Rate. 131–135, 177, 178

CDnet2012 2012 Change Detection. v, vi, xvi, xix, xx, 6–9, 28, 39, 46, 47, 114, 117, 118, 127, 129, 132–138, 141–144, 169, 170, 173, 174, 177, 179, 180, 184–186

CDnet2014 2014 Change Detection. v, vi, xvi, xix, xx, 6–8, 28, 38, 39, 47, 115, 117, 118, 127–129, 132–137, 139, 141–144, 169, 170, 175–180, 184–186

CNN Convolutional Neural Network. 3, 5, 40, 47, 50, 53, 55, 56

CODD Custom object direction detection. 79, 82, 98–100, 108–112, 183

COTS Commercial-Off-The-Shelf. 66, 118, 145

CPG Central Pattern Generator. 71

CPU Central Processing Unit. xviii, 3–5, 57, 63, 64, 68, 69, 74, 75, 124, 144–149, 152, 154, 160, 162–164, 170, 179, 186, 187

CSNN Convolutional Spiking Neural Network. 26, 27

CUDA Compute Unified Device Architecture. 187

CV Computer Vision. 29

DAVIS Dynamic Active Pixel Vision Sensor. 65

DBS Dynamic Background subtraction. iv

DLT Deep Learning Tracker. 55, 56

DNN Deep Neural Network. 41, 47, 48, 56, 74, 76, 115, 116

DoG Difference-of-Gaussians. 20, 26, 84, 86

DSGC Direction-Selective Ganglion Cells. xiii, 5, 6, 78, 79, 83, 88, 182, 183

DSP digital signal processing. 171, 172

DVS Dynamic Voltage Scaling. 66, 67, 69

F1 F-measure. 131–135, 169, 177, 178

FN False Negatives. 130, 131

FNR False Negative Rate. 131, 132, 134, 135, 169, 177, 178

FP False Positives. 101, 111, 130, 131

FPGA Field-Programmable Gate Array. v, xvii, xviii, 4, 5, 7, 9, 57, 70–73, 75, 76, 143, 144, 146–150, 152–164, 171, 173, 174, 181, 182, 186–188

FPR False Positive Rate. 131, 132, 134, 135, 169, 177, 178

fps Frames Per Second. v, vi, 7, 59, 62, 128, 175, 177, 179, 180

GC Ganglion Cells. xiii, 15, 29, 83, 188

GLOH Gradient Location and Orientation Histogram. 52

GPU Graphical Processing Unit. v, 4, 57, 63, 68, 69, 75, 144, 146–149, 154–156, 186, 187

GSOC Google Summer of Code. xv, 6, 8, 9, 39, 41, 74, 98, 114, 115, 117–120, 127, 133, 134, 137, 140–142, 146, 161

HDD Hard Disk Drive. xviii, 163, 170, 179

HDL Hardware Description Language. 5, 70, 73, 144, 147, 152–154

HH Hodgkin & Huxley. 16–19

HLS High Level Synthesis. 70, 72, 73, 147, 152, 153

HMI Human Machine Interaction. 34

HOG Histogram of Oriented Gradients. 52

HSMD Hybrid Sensitive Motion Detector. iv, v, xv, xvi, xix, 6–9, 113–115, 118–120, 122, 124–127, 129, 130, 132–135, 137–144, 146, 149, 164, 166, 178–180, 184–187, 189

HSV Hue, Saturation and brightness Value. 46

IAF Integrate-and-Fire. 17–19

IC Integrated Circuit. 4

ICA Independent Component Analysis. 53

IEEE Institute of Electrical and Electronics Engineers. 152, 189

IIR Infinite Impulse Response. 43

IJCNN International Joint Conference on Neural Networks. 189

IPL Inner Plexiform Layer. 78

IZK Izhikevich. 18

KNN Mixture of Gaussians K Nearest Neighbours. 38, 98, 117, 118, 127, 134, 137

LAB logic array blocks. 150–152

LIF Leaky-Integrate-and-Fire. 17, 18, 20–22, 27, 28, 64–66, 119, 120, 125

LSBP Local Single Value Decomposition Binary Pattern. 39, 98, 117, 118, 127, 134, 137

MFCN Multiscale Fully Convolutional Network. 40

MHSNN multi-hierarchical spiking neural network. iv, xiii, 6–8, 51, 76–82, 92, 101, 108–114, 183, 184, 186, 189

ML Machine Learning. 3, 5, 41, 44, 63, 72, 74, 75

MLAB memory logic array blocks. 151, 171, 172

MOG Mixture of Gaussians. 37, 38, 48, 98, 117, 118, 127, 134, 137

MOG2 Gaussian Mixture Probability Density. 38, 98, 117, 118, 127, 134, 137

NDK NeuronHSMD device kernels. 162, 164, 166, 172, 173

NeREM Neural Response Mixture. 48

NeuroHSMD Neuromorphic Hybrid Sensitive Motion Detector. iv, v, xviii, 7, 9, 144, 147, 149, 153, 159–163, 169, 178, 179, 185, 187, 189

NHA NeuronHSMD Host Application. 162, 164, 165

NN Neural Network. 45, 46, 48, 64

NNA Neural Network Accelerators. 63–65

NoC Network-on-Chip. 155

NS Navigational Systems. 35

NUC Next Unit of Computing. 71

OMD Object Motion Detection. xii, 5, 6, 28–30, 32, 35, 40, 43, 44, 48, 56, 57, 62, 74, 75

OMS-GC Object Motion Sensitive Ganglion Cells. 5–7, 10, 15, 28, 41, 73, 114, 118, 119, 141–143, 146, 182, 185

OpenCL Open Computer Language. v, xvii, xviii, 5–7, 70, 72, 73, 144, 147, 149, 150, 153–163, 169, 187, 188

OpenCV Open Computer Vision. v, xviii, 8, 9, 39, 79, 98, 112, 114, 117, 118, 120, 124, 127, 129, 141, 162, 163, 183, 187

ORB Oriented FAST and Rotated BRIEF. 53

OS Operating Systems. 4

PCA Principal Component Analysis. xiii, 44, 45, 50, 53, 93–96, 108–111

PCC Percentage of Correct Classifications. v, 101, 108–112, 183, 186

PCIe Peripheral Component Interconnect Expres. xviii, 63, 68, 155, 156, 160, 163, 164

Pr Precision. 131, 132, 134, 135, 169, 177, 178

PS Processor System. 187

PTZ Pan, Tilt and Zoom. 128, 137

PWC Percentage of Wrong Classifications. 101, 104, 107–112, 183

R Average Ranking. xvi, xix, 131, 132, 135–140

RAM-NN Random Access Memory Neural Network. 45

RBCNN Region Based Convolutional Neural Network. 62

RBF-NN Radial Basis Function Neural Network. 45, 46

RBM Restricted Boltzman Machine. 47

RC Average Ranking across all Categories. xix, 131–135, 169, 177, 178

RC Resitance and Capacitance. xii, 18

Re Recall. 130–132, 134, 135, 169, 177, 178

ReSuMe Remote Supervised Method. xiii, 8, 27, 28, 67, 89, 90, 97, 98, 102

RF Receptive Field. 83–85

RGB Red, Green and Blue. 20

ROI Region of Interest. xvi, 129, 130, 133

RPCA Robust Principal Component Analysis. 44, 45

RSNN Recurrent Spiking Neural Network. 26, 27

RTL Register Transfer Level. 152

RTSS Real-Time Semantic Segmentation. 40

SAC starburst amacrine cells. xiii, 78, 79

SATA Serial Advanced Technology Attachment. xviii, 163, 164

SBS Semantic Background Segmentation. 40

SDAE Stacked Denoising Auto-Encoder. 47

SDK Software Development Kit. xvii, 154, 156–158

SDRAM Synchronous Dynamic Random Access Memory. 151

SIFT Scale-Invariant Feature Transform. 52, 53

SNN Spiking Neural Network. iv, v, 3–11, 20, 22–28, 41, 51, 63, 64, 66, 67, 69, 70, 72–76, 97, 112–115, 118, 119, 124, 125, 141–144, 146, 147, 149, 153, 155, 156, 160, 162, 177, 178, 180, 182, 184–187

SOBS Self Organizing Background Subtraction. 46

SOM Self Organizing Map. 43, 46

Sp Specificity. 130–132, 134, 135, 169, 177, 178

SRAM Static Random Access Memory. 71, 151

SSD Solid State Drive. xviii, 163

STDP Spike Timing Dependent Plasticity. 27, 28, 71

STL Standard Template Library. 124, 141

SURF Speeded Up Robust Features. 52

TN True Negatives. 130, 131

TP True Positives. 101, 130, 131

UAV Unmanned Aerial Vehicle. 62

USB Universal Serial Bus. xviii, 163, 164

VHDL Very High Speed Integrated Circuit Hardware Description Language. 70, 147, 152

VLSI Very Large Scale of Integration. 62, 75, 76

WCR Wrong Classifications Rate. 131–135, 177, 178

Chapter 1

Introduction

1.1 Background

In Computer Vision, object motion detection is the process of detecting moving objects relative to their surroundings [16]. Object motion detection is required in many real applications such as video surveillance of human activities, monitoring of animals, optical motion detection, multimedia application, Advanced-Driver Assist Systems (ADAS), and autonomous systems [16; 17]. There are several challenges (e.g. objects camouflage, illumination variation, motion blur) when performing object motion detection. Several object motion detection methods have been published over the years [16; 17], some methods are more accurate than others but are also more computationally intensive, while others are less accurate and computationally intensive. The majority of the object motion detection methods only focus on the accuracy of proposed methods solving a group of challenges and do not measure the speed [16; 17]. Nevertheless, there is a significant demand in terms of real-time robust object motion detection is required in applications such as autonomous systems' navigation, object tracking and changes detection [17].

The detection of moving objects in different directions and the motion detection of objects are done very efficiently by vertebrate retinas. Different retinal circuits trigger different functionalities such as light detection, motion detection and discrimination (i.e. interpretation of spatio-temporal patterns triggered by the retinal photoreceptors), object motion (i.e. detection of objects moving in the scene), identification of approaching motion (looming), anticipation, motion extrapolation, and omitted stimulus-response [18]. Vertebrate retinas are notable for i) incorporating millions of these retinal circuits, ii) being extremely efficient (the whole human brain consumes approximately 20 Watts) and iii) currently still displaying the capability to outperform any state-of-the-art computer [19].

1.2 Current approaches

The human brain is very efficient at performing computations, as it only takes about 25 watts for 86 billion neurons [20]. The brain computational efficiency is a consequence of massive parallelism. The retina, an extension of the brain itself, is responsible for performing the first stages of visual pre-processing, including the detection of movement [18; 21]. Although the detection of the directions described by moving objects and object motion detection are trivial tasks for vertebrate retinas [18; 21], these are still complex computational tasks using exiting Background subtraction (BS) methods [16].

On computers, the detection of movements is normally achieved through the use of BS methods. BS have been one of the most active research topics in computer vision and have been widely studied for the last 30 years [16; 22; 23]. [16; 22; 23]. In BS algorithms, object motion detection is obtained through the

extraction of the foreground (composed of moving objects) from the background (composed of static or semi-static movements) image. BS are used in applications such as intelligent surveillance of human activities in public spaces, traffic monitoring, industrial machine vision applications, etc [22]. More recently, BS algorithms have been improved using Machine Learning (ML) algorithms [16]. ML is a research field that focuses on the development of algorithms and methods for solving complex problems in a generic way [24]. These algorithms are characterised by learning the detailed design from a set of labelled data and learn a model or a set of rules from a labelled dataset so that the ML model can correctly predict the labels of data points in unforeseen datasets (i.e. datasets that were not used to train the model) [24]. Among other research topics, ML includes Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs) and Spiking Neural Networks (SNNs) [20; 23].

ANNs have been widely used for classification and pattern recognition tasks, but at the same time, ANNs lack biological plausibility [20]. Although CNNs are as recent as ANNs, their popularity has increased in recent years as a consequence of the evolution of the computational capabilities [22; 23]. CNNs are known by its capability to accurately classify objects and can be used for tracking of known objects but similar to ANNs ,they lack of biological plausibility [22; 23]. Unlike ANNs and CNNs that have been studied for more than thirty years, SNNs emerged about twenty years ago and have gained growing interest in researchers because of their biological plausibility. [20].

SNNs are well known for their biological plausibility, but also for the complexity inherited from biological systems, which are characterised by massive parallelism, according to [20]. Modern computation platforms rely heavily on CPU to provide compatibility and security with other devices/applications. Although

CPUs have been evolving in recent years, CPUs still relies on the von Neumann architecture proposed by John von Neumann proposed in 1945 [25; 26]. Blank [27] wrote in 2018 that Moore's law, which states that the number of transistors in dense Integrated Circuits (ICs) doubles every eighteen to twenty-four months, ended around 2008. Furthermore, the clock speed has reached technological limitations, preventing CPUs from working with frequencies above 4 GHz, which also introduces memory barriers, and power dissipation challenges [27]. Therefore, the design of the CPU paradigm has shifted into multicore and multiprocessor to overcome the limitations associated with the clock speed [26]. The scientific community agrees that the multicore and multiprocessor strategy will shortly meet technological limitations related to the increase in power consumption of these solutions. [26].

Neuromorphic engineering aims to develop hardware devices/platforms that mimic biologic nervous systems. Such neuromorphic solutions must be compatible with other devices and applications, which is a challenge because compatibility and security are provided via standard Operating Systems (OS) (e.g. Microsoft Windows, macOS and Linux) that only run on CPUs. The solution is heterogeneous computing, which combines CPUs with one or more processing technologies such as graphical processing units GPUs, FPGAs, or other Application-Specific Integrated Circuitss (ASICs) connected via high-speed external (e.g. PCIe) or internal (e.g. AXI and Avalon) buses [28; 29]. Heterogeneous computing and/or neuromorphic applications are ideal for hosting customised SNN.

1.3 Research gaps

The detection of the direction performed by moving objects and the Object Motion Detection (OMD) are used in many fields, including traffic and human activity monitoring, ADAS, object tracking, etc. Unlike computers, vertebrate retinas are highly efficient at sensing object motion and its direction. In this PhD research programme, the following gaps were identified:

- Modern BS and object motion detection algorithms use ML approaches such as ANNs and CNNs but are not suitable for real-time applications and lack biological plausibility.
- SNN are biologically plausible and can be used for implementing basic functionalities observed in retinal cells, such as in the Direction-Selective Ganglion Cellss (DSGCs) and Object Motion Sensitive Ganglion Cellss (OMS-GCs)
- SNN are massively parallel and therefore not suitable for being processed by CPUs
- FPGAs are specialised and flexible hardware devices that offer freely-reconfigurable logic, desirable for accelerating massively parallel algorithms
- Although FPGAs are normally reconfigured using Hardware Description Language (HDL) which are hard to master, OpenCL is a C-like programming language, it is simpler to master and more suitable for programming devices like CPUs and FPGAs

This research was built on the premise of the aforementioned gaps identified.

1.4 Aims and objectives

In challenging circumstances (e.g. low visibility, blurred vision, moving cameras with moving objects, occlusion, etc.), no BS or OMD outperforms vertebrate retinas. Is it possible to model or enhance existing object motion detection algorithms? Can such methods be accelerated using dedicated algorithms? The PhD research programme aimed to explore how object motion detection could be improved using SNN to mimic some of the DSGCs and OMS-GCs basic functionalities and accelerate SNNs using dedicated hardware.

The PhD research programme had the following objectives:

a) Research into DSGCs and OMS-GCs and replicate their basic functionalities (such as detecting horizontal and vertical movements and more generic object motions). The research lead to the development of the MHSNN architecture of a customised 4-layer SNN, inspired on DSGC available on vertebrate retinas, which is sensitive to vertical and horizontal movements. The results show that the MHSNN performed the correct detection of leftwards, rightwards, downwards, and upwards movements in 93.6%, 92.4%, 93.9% and 93.1%, respectively when tested against the custom semisynthetic dataset.

b) Explore the use of spiking neural networks to model and/or enhance existing object motion detectors. The HSMD algorithm combined an existing BS algorithm named GSOC, available on the OpenCL library, with a customised 3-layer SNN. The HSMD algorithm was ranked first overall against the CDnet2012 and CDnet2014 benchmark datasets against the competing BS algorithm. Furthermore, the HSMD did not introduce a substantial degradation of the frame rate, being capable of processing 720×480 at

13.82 fps (CDnet2014) and 13.92 fps (CDnet2012).

c) Optimise object motion detectors using FPGAs to improve the latency by accelerating SNN with minimal or no deterioration of the accuracy. The HSMD's customised 3-layer SNN was accelerated on an FPGA device using OpenCL and was tested against the CDnet2012 and CDnet2014 benchmark datasets. The NeuroHSMD has produced an acceleration of 82% over the HSMD and it is capable of processing 720 × 480 at 28.06 fps (CDnet2012) and 28.71 fps (CDnet2014).

1.5 Summary of the thesis

The PhD research programme focused on modelling a MHSNN for detecting horizontal and vertical movements, design of SNN to enhance an existing and efficient BS algorithm using a custom 3-layer SNN called HSMD and NeuroHSMD, which is the neuromorphic implementation of the HSMD algorithm, for optimising the computation speed of the custom 3-layer SNN. The thesis structure and a summary of each Chapter are presented below:

Chapter 2: Literature Review:

This chapter reviews the relevant common aspects to most vertebrate retinas' physiology, emphasising the OMS-GCs. This chapter also reviews the most relevant SNN works, highlighting their main advantages and disadvantages. The most relevant BS works, challenges, and limitations are also reviewed in this chapter. The Chap-

ter ends with a revision of relevant neuromorphic implementations of retinas and other brain functions.

Chapter 3 - Initial exploration of Spiking Neural Networks on motion detection:

The initial exploration of SNN lead to the development of a novel MHSNN architecture capable of detecting leftwards, right-wards, upwards, and downwards movements. It is also explained how the supervised learning ReSuMe was used to train the output layer neurons and how synapses with different propagation can be used to create local buffers. The MHSNN is capable of classifying simple movements using a semisynthetic dataset.

Chapter 4: HSMD - Hybrid Spiking Motion Detection:

The HSMD proposes the enhancement of the GSOC BS algorithm [30] available on the OpenCV library [31], and that has performed better on the CDnet2012 [9], and CDnet2014 [32] which are two of the reference datasets for benchmarking BS algorithms using customised 3-layer SNN. The 3-layer SNN was optimised to ensure that the HSMD algorithm could perform background subtraction

on the fly. The HSMD ranked first when benchmarked against all the BS available on the OpenCV library [31] using the eight metrics proposed in CDnet2012 [9].

Chapter 5: NeuroHSMD - Neuromorphic Hybrid Spiking Motion Detection:
A state-of-the-art NeuroHSMD is presented in this chapter. The 3-layer SNN used to enhance the GSOC algorithm, which was the bottleneck of the HSMD algorithm, was fully implemented on an Intel FPGA board [33] using OpenCL [34].

Chapter 6: Conclusions and Future Work

A summary of the work and achievements obtained during the PhD research programme is given in this chapter. A list of publications and future publications emerging from the results obtained in Chapter 5 is also presented in this chapter. The last section discusses future work and how the systematic methodology developed in this research programme can efficiently model other bio-inspired retinal cells.

Chapter 2

Background Research

This chapter reviews works relevant to the PhD research program. A brief introduction to the anatomy of the eye and retina is given in Section 2.1 to set the context and to introduce the OMS-GCs, which are the foundations of the PhD research programme. Spiking neuron models and SNN are discussed and in Section 2.2. Existing SNN simulators and their advantages/disadvantages are summarised in Section 2.3. Relevant works in terms of SNN architectures for image processing are analysed in Section 2.4. The reader will have the opportunity to understand the research gaps in terms of SNN architectures which justify the need for this PhD research programme. Challenges and use-case scenarios of object motion detection using classical computer vision methods are discussed in Sections 2.5. Neuromorphic and heterogeneous computing are analysed in Section 2.6 covers the importance of hardware implementations in the

acceleration of SNNs. The research gaps that led to this PhD research programme are discussed in section 2.7.

2.1 Anatomy of the Eye and Retina

The eye (see Figure 2.1) is a fundamental part of the vision system, and it is composed by the following parts: **Iris** which is the coloured part of the eye that is responsible for controlling the amount of light that reaches the retina that lays at the back of the eye; **Pupil** is a circular membrane in the centre of the Iris that dilates and contracts to increase or reduce the quantity of light that reaches the retina; **Cornea** is the transparent circular membrane covering the front of the eyeball which refracts the light to the Lens;

Lens is the transparent structure behind the Pupil that refracts the light into the retina; **Choroid** is the middle layer of the eye between the retina and the Sclera that contains pigments to absorb excess light to prevent image blurring; **Ciliary body** connects the Choroid to the iris; **Sclera** is the white and though part of the eye for maintaining the spherical configuration of the eye and offers resistance to internal and external forces; **Retina** a tiny tissue that lays at the back of the eyeball and is composed of thousands of neurocircuits responsible for converting the light intensities into visual information to be further processed by the brain;**Fovea** is a

tiny depression in the centre of the Macula (a yellow spot on the retina) and it is considered a highly-specialised region of the retina responsible for producing the sharpest vision with the highest colour discrimination; and **Optic nerve** which provides the multichannel connectivity between the retina and the brain.

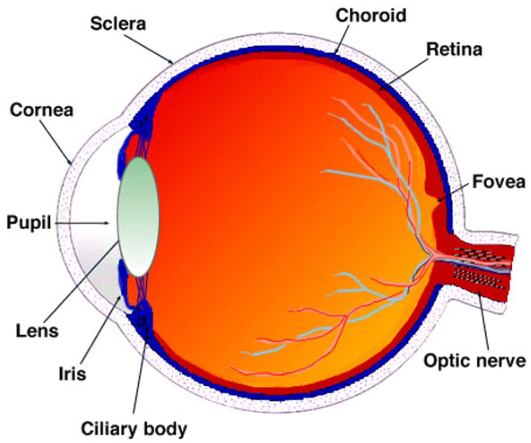

Figure 2.1: Anatomy of the eye. Adopted from [1].

The retina is considered an extension of the brain and has been widely studied since Cajal (1892) [35], [36]. Vertebrate retinas vary in type, shape, size, connectivity, and number of cell types. All vertebrate retinas are composed of photoreceptors (cones and rods), bipolar, horizontal, amacrine, and ganglion cells (Figure 2.2). Each of these types of cells comprises a wide range of functional subtypes [37].

Rods are sensitive to low intensity light, while cones sense light and colour. Cones can be subdivided into three categories based on

their ability to detect short, medium, and long light wavelengths [38]. Bipolar cells forward signals triggered by rods and cones to ganglion cells [38]. Horizontal cells are responsible for processing visual information received from bipolar cells, rods, and cones [1; 38]. Amacrine cells are responsible for modulating and integrating visual signals from bipolar, and ganglion cells [1; 38]. Ganglion cells receive visual information from bipolar cells and analyse shapes, contrast, motion, and colour, and forward them to the visual cortex via the optic nerve [1; 38].

Each retinal cell has a specific role in the vision system. The types of cells vary between animal species, and the number of retinal cells also varies according to the surrounding environment where each species lives. The organisation of the retina is shown in Figure 2.2

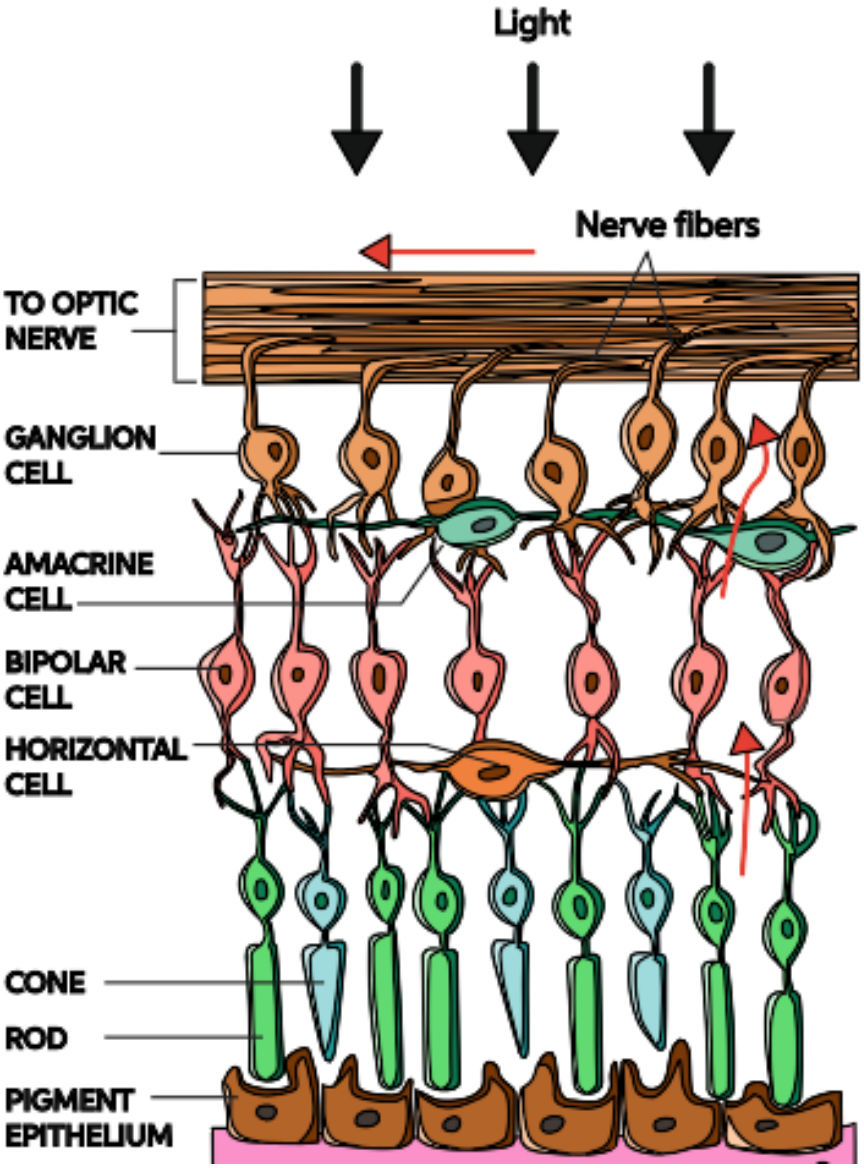

Figure 2.2: The structure of the retina comprises rods, cones, and horizontal, Amacrine, and ganglion cells. Light passes through all the retinal layers and hits the pigment epithelium responsible for protecting the outer retina from excessive light. Rods detect grey gradients and dim light, while cones detect red, green, and blue light. Horizontal cells regulate the visual information from rods, cones, and bipolar cells. Bipolar cells transport the visual information to ganglion cells. Amacrine cells regulate the visual information from bipolar and ganglion cells. Finally, ganglion cells receive and process the visual information from bipolar cells and transmit the post-processing information to the visual cortex via the optic nerve. Adopted from [2].

Object motion detection, light sensitivity, and looming (i.e. object approaching the eye) detection are three of the visual tasks that are observed in all vertebrate retinas, such as the light sensitive and looming GC. The OMS-GCs active respond when a local patch on the centre of receptive field moves with a trajectory different from the background; light sensitive GC respond to light intensity variations; and the looming GC respond to approach and recede motions [18]. More complex cells have been identified, including the fast response and the predictive GC. The fast response cells respond to fast motion variations, while the predictive cells trigger automatic responses to specific stimuli that were previously learnt [39]. The focus of this PhD research programme is to model and emulate the OMS-GC basic functionalities for improving existing motion detection algorithms, targeting a wide range of environmental conditions from dim light to challenging weather conditions.

2.2 Spiking Neuron Models

Models of the retina are generally either focused on single neuron cells, or on complex networks of neurons [40; 41]. Advances in retina research [7; 18; 42; 43; 44] have enabled researchers to understand better the anatomical and neurophysiological retinal function to de-

sign computational models that mimic some retinal aspects. Retinal responses are related to action potential initiation, dendritic processing, many levels of effects (such as ionic channels, physical properties, extracellular stimulation), motion detection, and motion anticipation [40]. Computational models have been used to augment the understanding of single neuron response dynamics and computation and their functional contributions in multi-hierarchy neural networks [40]. Typically, retinal models are modelled with a high level of abstraction, focusing on the individual action potentials [**?**]. The significant differences between approaches are the degree to which the authors can model the actual retina's behaviours [4].

In 1952, Hodgkin & Huxley (HH) [3] proposed the first biologically plausible and also the most computationally intensive spiking neuron model. The HH model consists of a semipermeable cell membrane that splits the internal cell, that behaves as a capacitor, from the extracellular fluids (see Figure 2.3) [3].

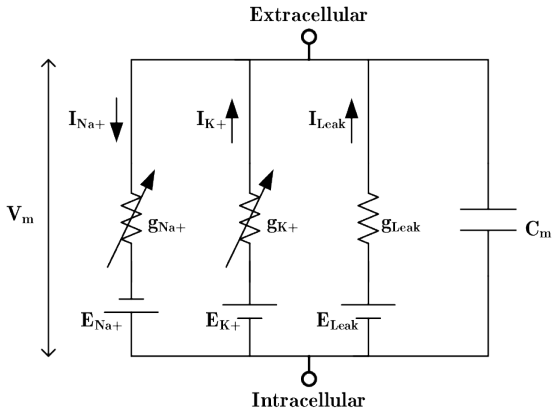

Figure 2.3: Schematic diagram of the Hodgkin-Huxley neuron model.V_m is the membrane potential and C_m is the membrane capacitance; I_{Na^+}, I_{K^+} and I_{Leak} are the currents associated to each channels. C_m is the membrane potential; g_{Na^+}, g_{K^+} are the non-linear electrical conductances that control the voltage-gated ion channels; g_{Leak} is the linear conductance. E_{Na^+}, E_{K^+} and E_{Leak} are the equilibrium potentials. Adopted from [3].

Currents are injected through the three different types of channels (sodium, potassium, and leakage), each with a distinct conductivity [3]. More specialised neuron models inspired on the HH and also based on differential equations were proposed such as the FitzHugh-Nagumo [45], Morris-Lecar [46], Hindmarsh-Rose [47],Komendantov-Kononenko [48] and Wilson [49].

A more simplistic model was proposed by Gerstner and Kistler [4] proposed the Integrate-and-Fire (IAF) and some variations, including the Leaky-Integrate-and-Fire (LIF). The main difference between the Leaky-Integrate-and-Fire (LIF) and IAF is that the LIF is more realistic than the IAF because it includes a leakage resis-

tor to lower the membrane potential voltage when no pre-spikes are received. LIF neurons can be represented using an electronic RC circuit, as shown in Figure 2.4.

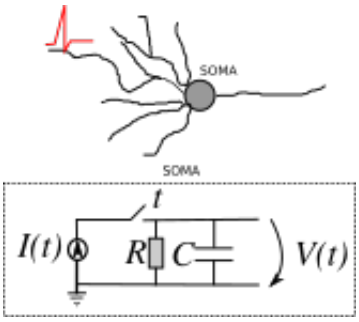

Figure 2.4: Schematic diagram of the leaky-integrate-and-fire neuron model. The base circuit is the module inside the grey circle on the right-hand side. A current I(t) charges the RC circuit. If the voltage V(t) across the capacitance reaches the threshold V_t then is a spike generated and V(t) is set to the reset voltage during a refractory period. Adapted from [4].

The action potential is governed by Equation 2.1.

$$\frac{\tau \delta V\left(t\right)}{\delta t} = -V\left(t\right) + RI\left(t\right) \tag{2.1}$$

where $\tau = RC$ is the time constant, R the membrane resistance, C the membrane capacitance, V(t) the membrane voltage at a given time t and I(t) is the current at time t.

Izhikevich [5] proposed the Izhikevich (IZK) model that is as biologically plausible as the HH model, but with a computational efficiency comparable to the IAF/LIF models. In his work, Izhike-

vich [5] presented a comparative study between some of the most relevant spiking and bursting neural models, discussing the use of each one in large-scale simulations. Figure 2.5 depicts the implementation of the computational cost versus biological plausibility of the spiking and burst neural models.

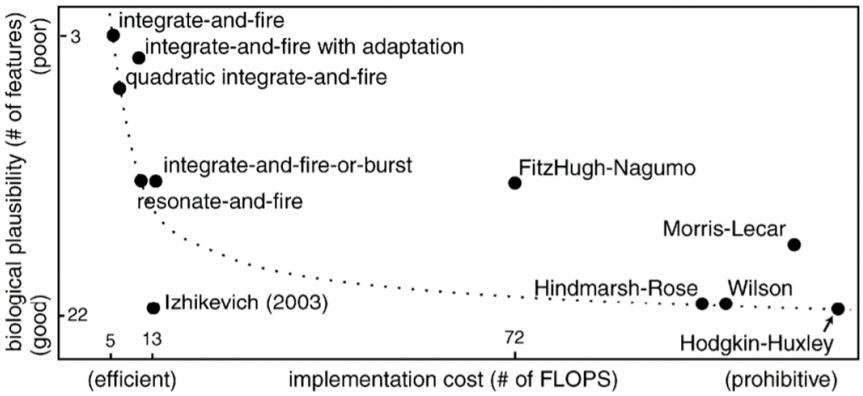

Figure 2.5: Spiking neural models and its performance. Adopted from [5].

Figure 2.5 shows that the simplest model, in terms of implementation cost, is the IAF. The IAF is also the least realistic neural model; however, some variations of IAF (IAF with adaptation, quadratic IAF resonate-and-fire and the IAF-or-burst models) have a higher level of biological realism [4; 5]. The Izhikevich model proves to have a good balance between biological plausibility and implementation cost [4]. Finally, the HH, FitzHugh-Nagumo, Morris-Lecar, Hinmarsh-Rose and Wilson which are the most realistic models, also require more floating-point operations per second

(FLOPS) [5]. The LIF neuron model has a good balance between processing cost and biological compatibility dynamics (see Chapter 3 and 4) and suitable for implementation in large-scale on (see Chapter 5 for further details). In this PhD research programme, the LIF model was selected over other Spiking Neuron models; because it requires less computational resources, which is desirable for implementing on dedicated hardware (see Chapter 5 for more details).

2.3 Spiking Neural Networks simulators

SNNs with different spiking neuron models and synaptic models, both parameterised with custom parameters and freely interconnected. The Retiner [50; 51] is a framework designed for testing the retina model, designed and implemented by the Cortivis consortium [52]. The Retiner simulates nine types of cells and includes edge detection, motion detection, and colour discrimination in real-time. The primary computational colours, Red, Green and Blue (RGB), and light intensity are filtered by Spatio-temporal retina-like filters Difference-of-Gaussians (DoG) and the laplacian-of-gaussian operators. The output of these filters is weighted and combined to produce a unique intensity matrix. A leaky integrator is used to convert the intensity matrix into a voltage activity matrix. The activity ma-

trix is then processed by LIF neurons, and the result is presented in the form of spike events.

Martinez-Canada *et al.* proposed the COREM computational framework for realistic retina modelling [6]. In this work, five computational retinal microcircuits were used as building blocks to model different retina mechanisms. The five computational microcircuits described in [6] are the space-variant Gaussian receptive field, low-pass temporal filter, single-compartment model, static nonlinearity, and short-term synaptic plasticity (see 2.6).

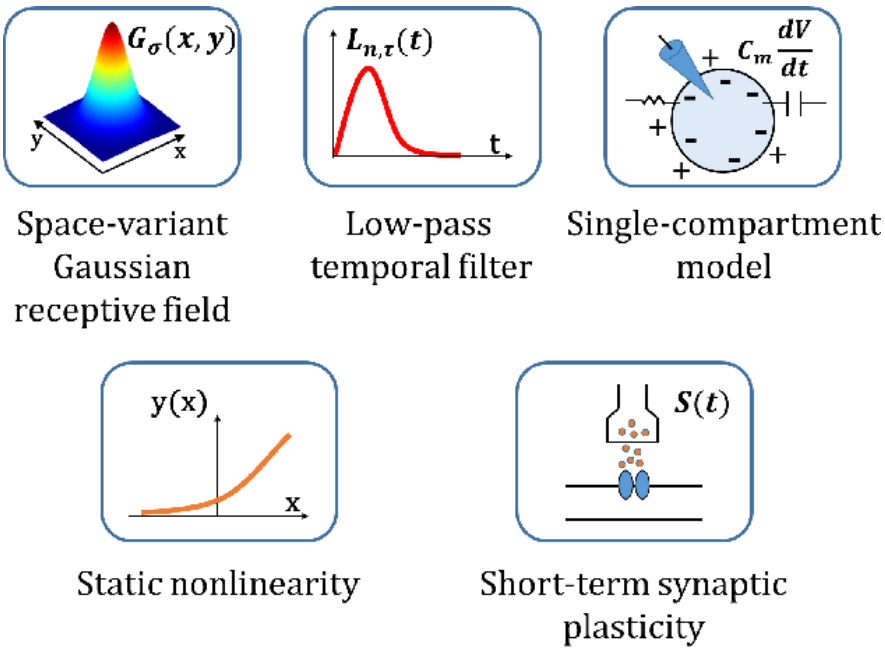

Figure 2.6: Computational retinal microcircuits that are used as basic building blocks within COREM. Adopted from [6]

The accuracy of the computational models was validated by fit-

ting published electrophysiological recordings. Martinez-Canada *et al.* modelled the adaptation to the mean light intensity, fast and slow temporal contrast adaptation, and the object motion-sensitive cells [6]. The five computational microcircuits, proposed by Martinez-Canada *et al.* [6], can be combined to reproduce single-cell and large-scale retina models at different abstraction levels; and can be classified as block-structured, block-compartment, or single-compartment models. The retinal model was then connected with NEST [53] to simulate ganglion cells using LIF neurons. Although Martinez-Canada *et al.* reported that it was possible to replicate retinal cell functionalities using COREM, neither accuracy nor speed details were provided [6]. The COREM framework functionality is represented in Figure 2.7.

Although the COREM and Retiner enable researchers to design Neural Networks, the three primary and widely used SNN simulators are Brian, NEURON and NEST [54]. Brian [55] is one of the most popular SNN simulators written in Python that delivers a user-friendly environment for users to write code quickly using vector-based computation to deliver efficient simulations. Furthermore, Brian provides flexibility to design custom SNN with spiking neurons freely interconnected between them. Brian simulator was replaced by the new Brian2 that delivers a more flexible and sim-

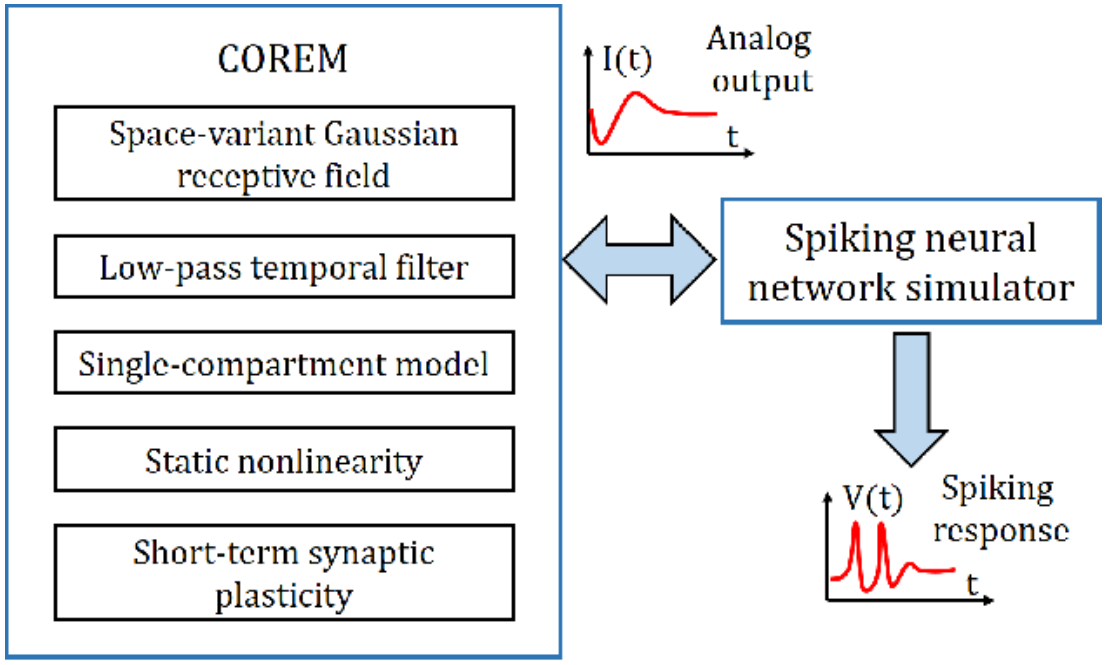

Figure 2.7: Interface of COREM with NEST. Adopted from [6]

plified environment for users to write new neuron models and more complex SNNs [56]. Brian 2 simulator which has been used in this research programme is described in Chapter 3. The neuron models are written as differential equations in standard mathematical notation [55]. NEURON is a SNN simulation toolbox that enables users to design, implement, and run efficient discrete-event SNN simulations, with easy to integrate hybrid simulations composed of spiking neuron models and cells with voltage gate conductance. The SNN simulator NEST [53] is also a simulation environment written in C++ with Python bindings, specially designed for models that

focus on the dynamics, size, and structure of SNNs and offers a wide range of built-in tools to monitor internal variable states and spike events. Many other packages and adaptors for Brian 2, NEURON and NEST, are available (such as PyNN, Cypress, NengoDL, etc.) but will not be discussed in this thesis; the reader can find more details in [54; 57; 58].

2.4 Spiking Neural Networks architectures suitable Computer Vision Processing

SNNs have been used for performing different tasks in Computer Vision. Long et al. [59] proposed the JSpike framework that delivers a catalogue of algorithms suitable for designing large-scale SNNs designed to require minimal memory and processing per synapse and therefore suitable for computer vision processing. The JSpike was one of the first SNN simulators optimised for computer vision processing. Nevertheless, there are no recent publications nor an active JSpike project, and therefore, it could be concluded that the JSpike project is inactive.

Wu et al. proposed bio-inspired SNN for segmenting objects and bind their pixels to construct object forms using excitatory lateral connections [60]. In a follow-up work, Wu et al. [61] proposed an-

other SNN for detecting moving objects in a visual image sequence. The SNN were trained for extracting the boundaries of moving objects from grey images [61]. Cai *et al.* [62] expanded the work in [60] and mimicked the basic functionality of motion detection with axonal delays. Despite the two SNN architectures [60; 62] being able to detect moving objects, neither is able to process moving objects in real-time.

Zylberberq et al. [63] used a biologically inspired sparse coding model using SNNs for emulating the types of responses that are found in cortical neurons in the primate visual cortex (also known as Gabor functions). Zylberberg et al. used a population of inhibitory neurons and a second population of excitatory neurons. The inhibitory neurons are used for producing lateral inhibition, which is required to generate the same patterns generated by Gabor functions. They also introduce a new unsupervised learning technique, an adaptation of Oja's learning rule, [64] for training the weights of both populations of neurons. Nevertheless, Gabor filters are especially useful in classification and recognition applications, which is not the main focus of this PhD research program.

Kerr et al. [65] proposed a custom SNN to model bio-inspired photoreceptors receptive fields and integrator neurons for extracting important features from intensity and range images, to perform

edge-detection. Although the bio-inspired edge-detection proposed by Kerr et al. perform this visual task, the authors did not perform an exhaustive analysis to assess the quality of edge detection. Furthermore, in 2015, Kerr et al. [66] proposed a four-layer hierarchical neural network for the extraction of complex features from natural images. The input image is convolved with DoG filters, and the result is converted by ganglion cells into spike events. The processing stages include edge detection, orientation detection, end-stopped detection and interest point detection. The proposed SNN was implemented in the Brian simulator [55] and is capable of processing 800×600 pixels images in 3.64s. Although 3.64s is a good processing speed for a SNN, this speed is far from ideal because the typical commercial camera frame rate is around 30 frames per second (33 ms).

Tavnaei et al. [67] who proposed a biological-inspired Convolutional Spiking Neural Network (CSNN) composed of a convolution layer, followed by a pooling layer and a fully connected layer (feature discovery). On classification tasks on the MNIST digit dataset[1], the proposed architecture showed an accuracy of 98% for clean (without any additive noise) images. Rueckert et al. [68] proposed a Recur-

[1]The MNIST database of handwritten digits is widely used to measure the accuracy of machine learning and artificial intelligence algorithms. http://yann.lecun.com/exdb/mnist/, last accessed: 27/02/2018.

rent Spiking Neural Network (RSNN) for planning tasks that detect movements. The proposed architecture is composed of 2 layers of LIF neurons, one for saving the current state and another for keeping the context. The RSNN is inspired by hippocampal neurons found in rats; however, such recurrence is not found in vertebrate retinas. Therefore, CSNN and RSNN are optimised to perform classification and not object motion detection.

Sun et al. [69] proposed a SNN that combines LIF neurons in hierarchical layers with excitatory/inhibitory pathways to describe receptive fields used to extract colour features for object recognition. Sun et al. made use of the unsupervised learning Spike Timing Dependent Plasticity (STDP) to train the excitatory synapses and therefore improve the classification accuracy on the 5 public datasets [69]. The authors claim an accuracy of about 90% in four of the 5 datasets used. Machado et al. [70] proposed the NatCSNN, a 3-layer convolutional spiking neural network for the classification of objects extracted from natural images. The authors [70] suggested the use of STDP unsupervised learning for training the middle layers and the ReSuMe supervised learning algorithm for teaching the output layer neurons using teacher signals. Although the NatCSNN scored an accuracy of 84.7%, the complex structure (use of a variation of the LIF with adaptive threshold neuron model for preventing over-

spiking, STDP and ReSuMe synapse models to interconnect neurons via many-to-many connectivities) of the NatCSNN introduces severe limitations in speed performance. Therefore, the NatCSNN is not suitable nor scalable for applications targetting real-time or near real-time processing. Although both works [69] [70] are based on hierarchical SNNs that use a combination of excitatory/inhibitory pathways to describe receptive fields, none of these architectures is easily scalable nor suitable for real-time applications.

The works proposed and covered in this section share the following similarities: (i) use of LIF neurons, organised in a hierarchical network; (ii) neural networks are composed of 3 or more layers which are interconnected via excitatory and/or inhibitory synapses, forming receptive fields; (iii) inclusion of final layer trained using STDP and/or ReSuMe. Neither of the studies addressed emulation of OMS-GCs and the reviewed SNN architectures are not easily scalable, nor are suitable for processing images in real-time. Although only the SNN architecture that was proposed by Wu et al. [61] was capable to perform OMD, the proposed SNN was not thoroughly tested against any public OMD datasets (e.g. CDnet2012 [9] or CDnet2014 [32]). Therefore, this PhD research programme explored the use of SNN for performing OMD in real-time applications.

2.5 Object Motion Detection

The detection of moving objects from video frame sequences is a trivial visual task performed by vertebrate retinal GC [18; 21] and yet a challenge in the Computer Vision (CV) research field. OMD is one of the most researched fields in computer vision and has been studied for more than 30 years [16]. OMD in videos captured from static and/or moving cameras is essential for a wide range of computer vision applications such as video surveillance, object collision avoidance, ADAS, etc [16; 17; 71]. Although the initial OMD models were designed for static cameras, the advances in sensor technology and the accessibility to portable devices fitted with cameras is triggering more challenging scenes where both cameras and objects can move at the same time [16]. OMD includes the following tasks: 1) Background subtraction, 2) noise reduction, 3) threshold selection and 4) moving objects detection (see Figure 2.8).

Several challenges have been identified in various works [16; 17; 22; 72; 73; 74] and can be summarised as follows :

- **Bootstrapping**: the sequence of images includes objects in both the background and foreground.
- **Camouflage**: the objects in the foreground are either obstructed by background objects or are composed of similar

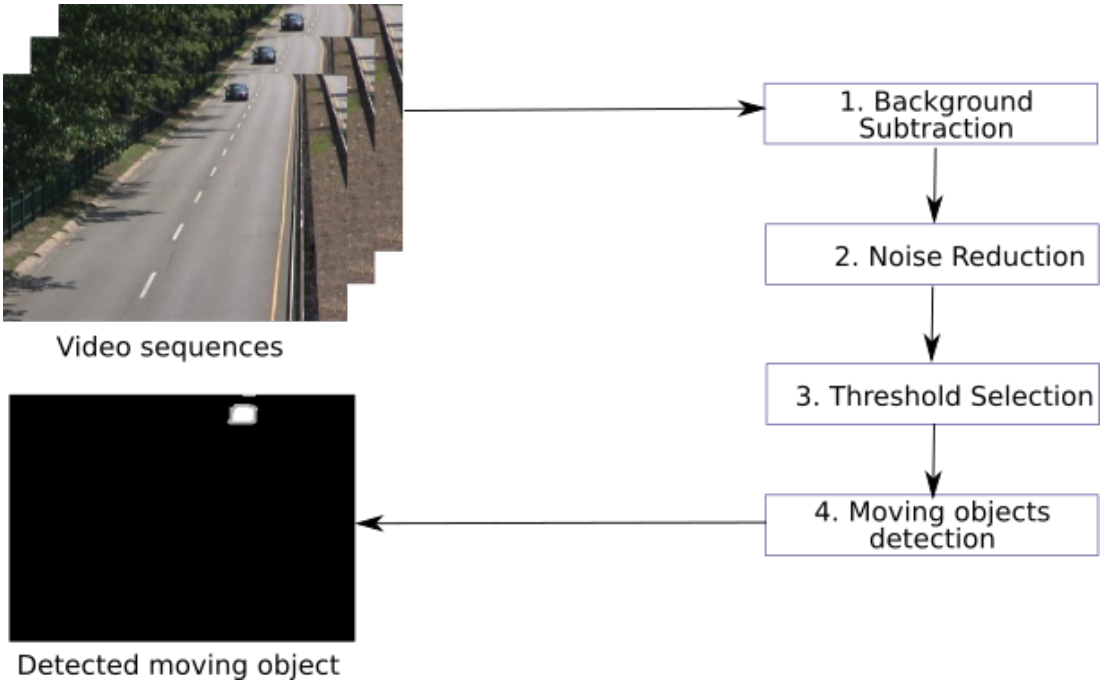

Figure 2.8: Object Motion Detection steps.

colours.

- **Dynamic background**: the objects in the background include parasitic movements such as surface water movement, branches and leafs shaking in trees, flags on windy days, etc.
- **Camera aperture**: blurred background and foreground as a consequence of the incorrect opening in a lens through which light passes to enter the camera.
- **Variation of illumination**: instant variations of illumination will increase the number of false-positive detections (i.e. pixels that should belong to the background are classified as

foreground).

- **Low frame rate**: the temporal distance between image frames prevents instant updates of the background and illumination changes, which reduces the accuracy and increases the number of false positives.
- **Motion blur**: caused by rapid camera movements or jittering, which blurs the image.
- **Parallax**: the apparent displacement of an object as a consequence of the camera movement. The parallax will have implications on the background modelling and its compensation.
- **Moving camera**: moving cameras introduce complexity because the static objects seem to be moving, and objects moving at a similar speed in the same direction of the camera will seem to be static.
- **Background objects movement**: although static objects can be added to and removed from the background, such objects should still be considered static.
- **Night videos**: night videos have dim light, lower contrast and reduced colour information.

- **Noisy images**: low-quality sensors, dust exposure, dirty lens, bright lights and low resolution are examples of factors that cause noisy images.
- **Shadows**: shadows created by objects when exposed to light sources (e.g. sun rays and artificial illumination) should not be part of the foreground models.
- **Stationary foreground objects**: a foreground that has stopped moving for a short period should not become part of the background model;
- **Challenging weather**: weather conditions (such as fog, rainstorms, strong winds, intense sun rays) have a major impact on the image quality and reduce the quality of the image drastically.

Furthermore, Garcia et al. [17] and Chapel et al. [16] identified OMD use-cases that can address many challenges discussed above and can be summarised as follows:

- **Visual analysis of human activities** fixed, or movable cameras used for monitoring human activities. Human activities can include highway maintenance (e.g. traffic density estimation, vehicle tracking, detection of dangerous manoeuvres)[75; 76; 77; 78; 79; 80; 81; 82]; tracking people in public places (e.g.

airports, train stations, seaports) [83; 84; 85; 86]; monitoring specific people/objects in mass events (e.g. sports games, music concerts, manifestations, and gatherings) [87; 88; 89]; and indoor/outdoor behaviour analysis (e.g. track people in closed public spaces, body-cameras installed police forces) [90; 91; 92].

- **Visual observation of animals behaviours** the observation of animals enables one to better understanding the health status of individual and/or colonies of animals. Animals behaviours may include monitoring of livestock (monitor cows, pigs, and diseases detection through the analysis of atypical movements) [40; 93; 94; 95]; gathering understanding about complex colonies of insects (e.g. bees and ants have communication mechanisms that enable them to work together to solve complex problems) [96; 97; 98; 99]; and monitoring wildlife (e.g. track movements of shoals of fish or pods of whales) [100; 101; 102; 103; 104; 105].

- **Visual observation of natural environments** the detection of foreign objects in natural environments is crucial for such environments. Some examples of natural environments are forests, lakes, rivers, oceans, and glaciers that require active human intervention to protect biodiversity in terms of fauna

and flora [106; 107; 108; 109; 110].

- **Visual hull computation** object motion detection is currently being used in many sports (such as Football, Tennis, and Athletics) to perform athletes movements analysis [111; 112; 113; 114; 115; 116].

- **Human Machine Interaction (HMI)** gesture recognition enables users to interact with machines. HMI is currently being used in many fields such as in games (gestures are mapped into game instructions); health (capture of facial motions to enable patients with severe muscular degenerative diseases to communicate using computers); and augmented/virtual reality (enable interaction between users and mixture of real and virtual objects for completing specific tasks) [117].

- **Content-based video coding/decoding** the foreground can be extracted from the background and encoded and streamed to the destination. At the destination, the foreground can be decoded and added to the pre-existing background model [118; 119].

- **Background substitution** state-of-the-art conference platforms enable users to blur or substitute the background in conference calls. The background substitution is being widely used

to protect users' privacy [120; 121].

- **ADAS and Navigational Systems (NS)** require constant updates of the foreground models to ensure that autonomous/semi-autonomous systems can safely navigate without colliding with a multitude of objects and living beings moving at random speeds and variable trajectories [122; 123; 124].

2.5.1 Background Subtraction

Several surveys about BS have been published in the literature focused on static or semi-static (i.e. cameras fixed in a given position exhibiting pan-tilt-zoom movements) scenes. McIvor [125] published, in 2000, one of the first OMD surveys where nine BS methods which were only described in detail but not compared. Piccardi [126] presented, in 2004, a comparative study between seven algorithms cording to speed, memory resources utilisation and accuracy (see Table 2.1). Piccardi's study [126] aims to facilitate the BS selection based on speed, memory requirements and accuracy requirements.

Cheung et al. [136] proposed a method for validating foreground regions (blobs) using a slow-adapting Kalman filter and compared the proposed method against six other methods using the recall and precision metrics. Elhabian et al. [137] covered several back-

Table 2.1 BS methods and their performance analysis

Method	Speed	Memory	Accuracy
Running Gaussian average [127; 128]	high	low	acceptable
Temporal median filter [129; 130]	high	low	acceptable
Mixture of Gaussians [131]	low	high	very good
Kernel density estimation [132]	low	high	very good
Sequential kernel density approximation [133]	low	acceptable	good
Cooccurence of image variations [134]	acceptable	acceptable	good
Eigenbackgrounds [135]	acceptable	acceptable	good

ground removal algorithms and identified that all the BS algorithms follow four significant steps, namely, pre-processing, background modelling, foreground extraction, and validation. Although the review was very comprehensive, the focus was on recursive and non-recursive approaches, which are suitable for background maintenance but less suitable for background modelling. Cristiani et al. [138] reviewed BS methods that can be applied to data captured from different sensor channels (including audio). Elgammal [139] reviewed more than 100 papers about object motion detection for static and moving cameras, highlighting the challenges and suggesting which method to use in each case. Bouwmans et al., Garcia et al. and Chapel et al. published comprehensive surveys [16; 17; 22; 72; 73; 74] focusing on traditional, recent, and prospective object motion detection methods.

Consecutive frame difference, background modelling and optical flow are the main categories for BS. Consecutive frame difference

methods are the simplest to implement and require less computational resources, but are also the most sensitive to the challenges listed above [16; 17]. In contrast, optical flow methods are the most robust but require more computational resources and, consequently, are not suitable for real-time applications [16; 17]. Therefore, background modelling methods are commonly used methods for extracting the foreground from the background in real-time applications [16; 17].

BS generic steps are detailed in Figure 2.9.

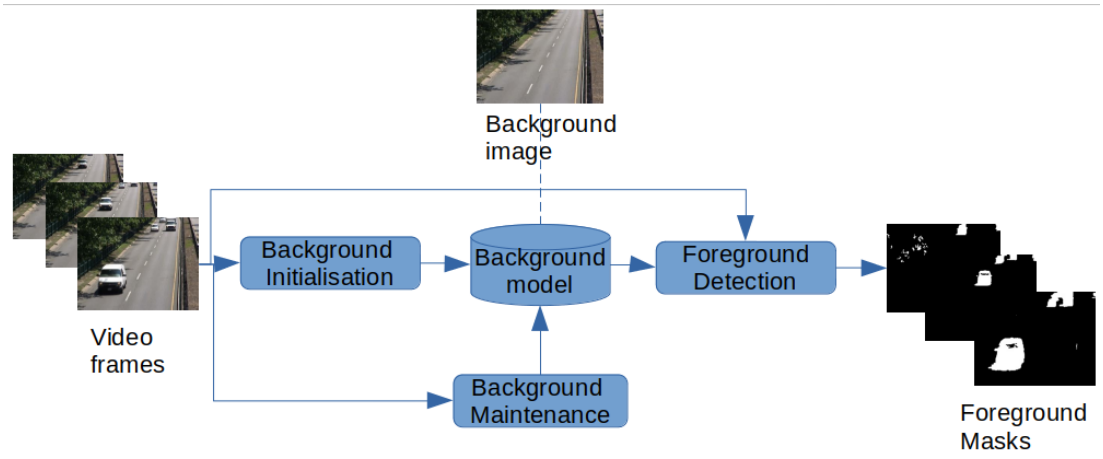

Figure 2.9: Background subtraction steps.

Stauffer & Grimson [131], and KaewTraKulPong & Bowden [140] suggested modelling each pixel as a Mixture of Gaussians (MOG) where the Gaussian distributions of the adaptive mixture model are analysed for determining which ones are likely to belong to the

background process. All the pixel values that do not fit in the background distributions are considered foreground [131]. Zivkovic [141] proposes an efficient adaptive algorithm using the Gaussian Mixture Probability Density (MOG2) for enhancing the MOG algorithm. MOG2 selects automatically the number of components per pixel, which results in complete adaptation to the observed scene. Zivkovic & Heijden [142] identified recursive equations for updating the parameters of the MOG and proposed Mixture of Gaussians K Nearest Neighbours (KNN) for the automatic selection of the pixel components. The Gaussian mixture based algorithms (MOG, MOG2 and KNN) show good robustness when exposed to noise and losses due to image compression but lack sensitivity to intermittent object motion, moving background objects and abrupt illumination changes.

In 2016, Sagi Zeevi [143] proposed the CNT algorithm, which performed better on the CDnet2014 dataset [32] and targets embedded platforms (e.g. Raspberry PI[1]). The CNT uses minimum pixel stability for a specified period for modelling the background; this can vary from 170 ms, default for swift movements, up to hundreds of seconds, where 60s is the default value [144]. Godbehere *et al.* [145] suggested a single-camera statistical segmentation and tracking al-

[1]Available online, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, last accessed 28/03/2022

gorithm named the GMG by combining per-pixel Bayesian segmentation, a bank of Kalman filters, and Gale-Shapley matching for the approximation of the solution to the multi-target problem. The proposed GMG algorithm is limited when processing video streams susceptible to camouflage, losses due to image compression, and noise.

Guo *et al.* [146] reported an adaptive Background subtraction model enhanced by a local Local Single Value Decomposition Binary Pattern (LSBP) for addressing illumination changes. The proposed LSBP algorithm enhances the robustness of the motion detection to illumination changes, shadows, and noise. However, the LSBP is less effective when processing video streams susceptible to camouflage, losses due to image compression, or external noise. More recently, in 2017, OpenCV released an improved version of the LSBP algorithm, also known as GSOC [147; 148], developed during the Google Summer of Code event [149], which enhances the LSBP algorithm by using colour descriptors and stabilisation heuristics for motion compensation [30; 148]. The GSOC algorithm demonstrates better performance on the CDnet2012 [9], and CDnet2014 [32] datasets [30; 31] when compared to other algorithms available on the OpenCV library.

More recently, Braham et al. [150] proposed a Semantic Background Segmentation (SBS)that uses object-level semantics to meet a range of problematic background subtraction conditions. The proposed SBS reduces false positive detections by integrating the output information of a semantic segmentation method, expressed as a probability for each pixel, with the output of existing BS methods. Inspired by Braham's work [150], Zeng et al. [151] proposed a Real-Time Semantic Segmentation (RTSS) for performing BS. The RTSS consists of two components: a BS segmenter B and a semantic segmenter S that work in parallel for foreground segmentation. The RTSS achieves state-of-the-art performance among most unsupervised background subtraction methods while functioning in real-time as compared to other BS methods [151]. Liang et al. [152] proposed a deep background subtraction method using a directed learning strategy that learns a specific CNN model for each video without manually labelling. Zeng et al. [153] proposed a Multi-scale Fully Convolutional Network (MFCN) architecture for background subtraction that takes advantage of diverse layer features. The deep features learned from MFCN improves foreground detection and that the complexity of the background subtraction process can be easily handled during the subtraction operation itself.

BS is the first step and the most important step of the OMD.

BS algorithms are responsible for extracting the foreground from the background using the spatio-temporal information of the light variation between sequential video frames. Furthermore, BS can be done using methods based on signal processing, ML, Deep Neural Network (DNN) or mathematical models. Although signal processing, ML and DNN tend to exhibit better accuracy than mathematical models, these algorithm types are also computationally intensive, introducing undesirable latencies. Nevertheless, mathematical models exhibit lower accuracy but require fewer computational resources and, therefore, are suitable for real-time applications. Moreover, the methods proposed by Braham et al. [150] and Zeng et al. [154] demonstrated that existing BS algorithms can be improved when combined with semantic segmentation models. Therefore, this PhD research programme combined a customised SNN for improving existing GSOC BS algorithm for mimicking simple biological functionalities observed in OMS-GC of vertebrate retinas.

2.5.2 Noise reduction

Parks & Fels [155] evaluated several noise reduction post-processing strategies and concluded that morphological operators outperform other techniques such as the median filter. This technique can re-

pair small gaps in the foreground segmentation and erase small clusters of pixels that were incorrectly labelled as foreground. Shadows and reflections introduce severe challenges, since most moving object recognition techniques incorrectly classify their region pixels as moving objects. Although it is preferable to deal with these issues by developing an effective algorithm at the time of detection, post-processing methods based on threshold selection can be useful in removing erroneous information from the output data. Solehah et al. [156] proposed comparing the current image's histogram with the one of the warped background and thresholding it to re-classify the pixels to remove noisy pixels associated with the foreground. This PhD research programme used median filters for filtering salt-and-pepper noise [157] as consequence of random pixel illumination variations.

2.5.3 Threshold selection

The threshold can be set to the same value for all pixels and the series. This plan is straightforward, but it is not ideal because pixels depict various actions, necessitating the use of an adaptable threshold. This can be accomplished by determining the threshold using the local temporal standard deviation of intensity between the background and current images, and then updating it with an

Infinite Impulse Response (IIR) filter as suggested by Collins et al. [158]. According to Wren et al. [159], an adaptive threshold can also be calculated statistically from the pixel variance. Chacon-Muguia and Gonzalez-Duarte [160] proposed the use of fuzzy thresholds for performing adaptive thresholds using a one-to-one Self Organizing Map (SOM) architecture to deal with dynamic backgrounds for object detection and shadow removal. This PhD research programme used populations of spiking neurons (see Chapters 3 and 4) and synapses with different propagation delays (see Chapter 3) to perform adaptive threshold selection.

2.5.4 Moving object detection

Detecting the physical movement of an object in a specific location or region is known as OMD. Unlike BS algorithm, which compares light variation between the current frame and previous frames, the OMD detects moving objects in the scene. Moreover, the quality of the BS (covered in section 2.5) has a direct impact on the quality of the OMD which is the building-block of other advanced visual tasks (such as classification of moving objects, tracking, interpretation and description of actions, human identification, or fusion of data from multiple cameras) [161]. This PhD research programme was focused on the OMD and therefore, this section will briefly review

some of the most relevant OMD methods available in the literature.

2.5.4.1 Representation learning

Modelling the background is a complex visual task, especially in scenarios where the camera, objects, or both are moving. Therefore, ML/Artificial Intelligence (AI) methods are normally used for modelling the background [16]

Oliver et al. [135] proposed the use of PCA models to lower the dimensionality and perform unsupervised learning of the illumination variation, which is referred to as subspace learning. Other subspace learning methods based on discriminative [162; 163] and mixed [164] models were proven to be more robust for foreground detection. Nevertheless, PCA based models are very sensitive to noise, outliers and incomplete data.

Robust Principal Component Analysis (RPCA) with decomposition into low-rank plus sparse matrices have been frequently employed in the area to address the PCA based algorithms limitations [165; 166; 167; 168]. Although these approaches are tolerant to light variation as well as dynamic backgrounds, batch algorithms are required, which makes these methods unsuitable for real-time applications [16]. Dynamic RPCA and robust subspace tracking have proven to overcome the RPCAs' limitations and attain real-

time performance of RPCA based algorithms [169; 170; 171; 172; 173; 174; 175].

Tensor RPCA based methods [176; 177; 178; 179] allow for the consideration of spatial and temporal constraints, making them more noise resistant. Although tensor/dynamic RPCA based algorithms are more robust than PCA based algorithms, they are also computationally intensive. Therefore, PCA was used in the work presented in section 3 for reducing the dimensionality of the datasets' image frames.

2.5.4.2 Neural networks modelling

Schofield et al. [180] were the first to suggest a Neural Network (NN) for background modelling and foreground detection named Random Access Memory Neural Network (RAM-NN). RAM-NNs are trained with a single pass of background images, which introduces limitations related to images having to accurately reflect the scene's background and no background maintenance stage. Tavakkoli [181] proposed a NN for separating the background into chunks during the training process. Radial Basis Function Neural Network (RBF-NN) are trained with background samples corresponding to its associated block. RBF-NN performs as a detector rather than a discriminant, generating a near border for the known class. Furthermore,

RBF-NNs can learn the dynamic background while addressing dynamic object detection as a single class problem, but also requires a large dataset to be able to generalise the background scenario. Maddalena and Petrosino [182; 183; 184; 185] proposed the Self Organizing Background Subtraction (SOBS) method, which is based on a 2D self-organizing NN designed for preserving pixel spatial relationships. The background is automatically modelled using the network's neuron weights. A neural map with $n \times n$ weight vectors is used to represent each individual pixel. The Hue, Saturation and brightness Value (HSV) colour space is used to initialise the weight vectors of the neurons with the relevant colour pixel values. For each new image, each individual pixel value is compared to its current model to identify whether the pixel relates to the background or the foreground. Several SOBS-based variations have emerged including (multivalued SOBS [186], SOBS-CF [187], SC-SOBS [188], 3dSOBS+ [189], Simplified SOM [190], Neural-Fuzzy SOM [191], and MILSOBS [192]), allowing it to remain among the top methods on the CDnet2012 [9]. SOBS also performs well in the detection of stationary objects [193; 194; 195]. However, one of the major drawbacks of SOBS-based approaches is that at least four parameters must be manually adjusted, and these methods lack biological plausibility. Therefore, the NN modelling methods were not used in

this PhD research programme.

2.5.4.3 Deep Neural Network modelling

DNNs have been used for performing moving object detection for moving cameras due to the availability of large annotated video datasets such as the CDnet2012 [9] and CDnet2014 [32]. Babaee et al. [196] proposed a single CNN that learns relevant features from data and predicts an appropriate background model from video. The proposed method can be utilised in applications for the detection of moving objects in a variety of video scenarios. The single CNN method proposed by Babaee et al. [196] was followed by improved versions that address: foreground detection enhancement [151], ground-truth generation [197], and learning of deep spatial features [198; 199; 200]. The methods above-mentioned are very accurate and address different artefacts, but they are not suitable for real-time applications.

Guo and Qi [201] and Xu et al. [202] proposed the use of Restricted Boltzman Machines (RBMs) łmodelling the background creation to achieve moving object detection. Xu et al. [203; 204] used deep auto-encoder networks to accomplish the same goal, whereas Qu et al. [205] used a context-encoder for background initialization. Zhang et al. [206] proposed the Stacked Denoising Auto-Encoder

(SDAE) to learn robust spatial features and density analysis to model the background, whereas Shafiee et al. [207] proposed the Neural Response Mixture (NeREM) to acquire deep features for improving MOG model proposed by Stauffer & Grimson [131], and KaewTraKulPong & Bowden [140]. Although DNNs have proven to perform object detection with a good accuracy, these types of NN lack from biological plausibility and therefore, such methods were not used in this PhD research programme.

2.5.5 Advanced Object Motion Detection applications

OMD methods are the building blocks for real-time applications such as trajectory classification and object tracking [208]. Although most OMD methods work offline by analysing recorded video sequences, the need for real-time moving object detection is increasingly being recognised, particularly in fields such as sociology, criminology, suspect behaviour detection and tracking, traffic accident detection, crowd tracking, and vehicle and robot navigation [208]. In general, the necessity for a real-time system necessitates a very short calculation time as well as minimal hardware and memory requirements [208].

2.5.5.1 Trajectory classification

Several works have used trajectory classification to find moving objects in video sequences acquired by cameras [16; 208]. The trajectory classification generic steps (see Figure 2.10) include i) selection of the initial point of interest corresponding to the moving object's centre of mass in the first image frame, ii) tracking the progression of the point of interest, and iii) classification of the trajectory described by that point of interest.

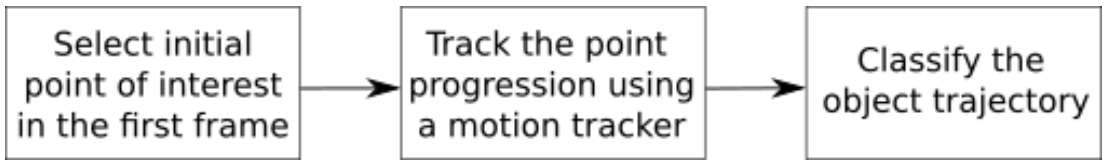

Figure 2.10: Trajectory classification steps. i) selection of the initial point of interest corresponding to the moving object's centre of mass in the first image frame, ii) tracking the progression of the point of interest and iii) classification of the trajectory described by that point of interest.

Sheikh et al. [209] employed a scale space for the background that was formed by the fundamental trajectory bases, and then classified trajectories that were not in this area as belonging to moving objects. Although the proposed method relied just on 2D picture quantities, the homography limitations were successfully extended to 3D scenes to accommodate freely moving cameras. While the algorithm is computationally expensive, the findings revealed that it performs well [209].

Brox & Malik [210] proposed the use of optic flow for compensating long term motion in a video sequence and extracting the trajectory points. Furthermore, spectral clustering was used to classify background and foreground trajectories. The approach can handle a high number of sequenced frames and partially occluded objects, but it fails to segment the objects densely. Yin et al. [211] used optic flow to compensate for camera parasitic movements, and subsequently PCA was employed to reduce the number of unusual trajectories. Moreover, the watershed transforms to distinguish foreground and background trajectories. Although label inference is used to handle unlabelled pixels, it fails to create dense segmentation of moving objects [211]. Singh et al. [212] addressed trajectory detection of freely moving cameras in footage acquired while the camera was worn by a human. Instead of employing elaborate models, they use point tracking with optical flow and a bag of words classifier to discern the trajectory of moving objects in their research. The proposed method works well for recognising first-person actions [212]. Zhang et al. [213] proposed a pre-trained CNN to recognise moving object trajectories in an unconstrained video over time by learning adaptive discriminating characteristics from short video clips extracted from the main video. Local tracklets (short trajectories) are extracted from each video clip and linked together across

sequential video clips. Even with unrestricted camera movement, the proposed method [213] works effectively for multi-face tracking. However, depending on the nature of moving objects, the proposed method requires wider training data [213].

In this PhD research programme, a custom SNN architecture is proposed for trajectory classification (see Chapter 3). The proposed MHSNN architecture combines excitatory and inhibitory synapses with different propagation delays for detecting the object trajectory.

2.5.5.2 Object tracking

Tracking is the process of locating a moving object in sequences of frames [208]. The object tracking generic steps (see Figure 2.11) include: i) selection of the target moving object, ii) store the moving object features, iii) extract moving objects features in the current frame, iv) select the best set of features that matches the target moving objects and v) update the moving object features.

The selection of the target moving object can either be done manually by the user or automatically using object detection methods using saliency detection or object segmentation and recognition from a single image [208]. Colours, texture, edges, geometric information, frequency coefficients, simple pixel grey values, or a combination of all of these object properties that comprise a feature

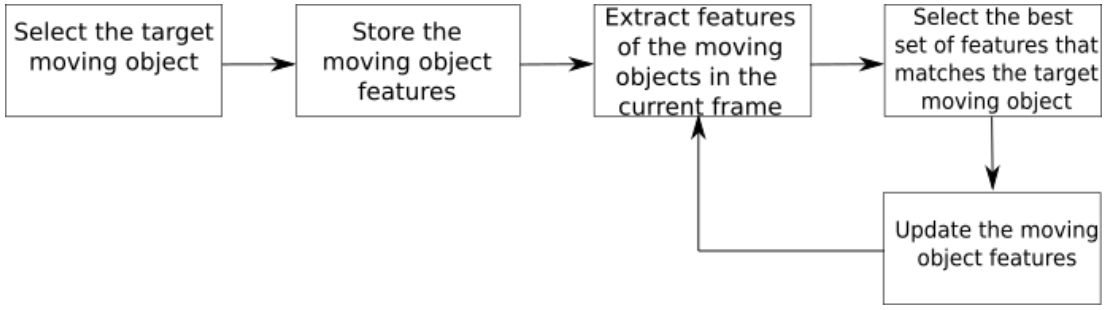

Figure 2.11: Object tracking steps. i) selection of the target moving object, ii) store the moving object features, iii) extract moving objects features in the current frame, iv) select the best set of features that matches the target moving objects and v) update the moving object features

space [214; 215; 216; 217; 218] can be used to characterise the target moving object. Other features for appearance modelling include the colour histogram [219] and the Histogram of Oriented Gradients (HOG) [220] to use a set of local histograms to characterise a moving object. Furthermore, the occurrence of gradient orientation in a local region of the object is counted in these histograms. Although HOG provide strong local information about a moving object, it is susceptible to variations in illumination.

Local feature descriptors are crucial in the image registration process and have a direct impact on the accuracy and robustness of image registration [221]. Some of the most common local feature descriptors algorithms utilised for representing moving objects are Gradient Location and Orientation Histogram (GLOH) [222], Speeded Up Robust Features (SURF) [223], Scale-Invariant Feature

Transform (SIFT) [224], Binary Robust Independent Elementary Features (BRIEF) [225] and Oriented FAST and Rotated BRIEF (ORB) [226]. SIFT is particularly desirable since it produces local features that are resistant to scale, noise, illumination, and local geometric distortion [227]. Nevertheless, SIFT's object tracking speed degrades as the number of falsely matched key points increase, which is not desirable when monitoring small objects [227]. CNNs can be used for learning adaptive, hierarchical, and distributed features. Such features can be used to update the object's appearance in real time [228] and perform descriptors encoding local spatial-temporal properties [229].

Statistical models can be used to match the target moving object features to the features extracted from the current frame [208]. Subspace techniques such as PCA and Independent Component Analysis (ICA) can be utilised to decrease the dimension of the feature space. The target moving object's location in the current frame should be determined using a matching strategy. The similarity/correlation function will select the best moving object candidate in the current frame that is the most similar to the target moving object based on appearance features taken from both the candidates and the target [208]. The target moving object features are updated after the target is detected in the current frame, and the process re-

peats for the following frames [208]. Ning et al. [230] propose using the statistics of the object's visual features (colour and texture) for the matching stage. Leal et al. [229] proposed a template-based method that uses a simple geometric shape, contour, or silhouette of the moving object. Nevertheless, the proposed basic template-base approach fails when the pose of the moving object changes. Pan et al. [231] proposed the Content-Adaptive Progressive Occlusion analysis (CAPOA) method that provides a clear distinction between the target and outliers by merging the information provided by spatio-temporal context and motion constraints together.

Comaniciu et al. [232] employed a kernel-based method using mean shift to construct a moving object's histogram based on appearance and similarity measurement using the Bhattacharyya distance to determine the best match for the moving object in the current and future frames. Bebenko et al. [233] proposed a learning model to identify pixel blocks containing the moving object, and track the moving object's behaviour in subsequent frames. Bagherzadeh & Yazdi [234] proposed the use of saliency maps to extract appearance cues in the frequency domain, and a regularised least squares classifier was used to classify pixels belonging to a moving object.

Another prominent approach for tracking moving objects is tracking-

by-detection [235; 236; 237] , which uses a detection algorithm combined with a matching technique to correlate the discovered items on sequenced images. This technique excels in difficult situations such as uncalibrated moving cameras, changing backgrounds, and, most notably, occlusion. However, matching step or object association is challenging, and the outputs are a distinct set of answers that frequently result in an increase of false positives. Chen et al. [238] and Sadeghian et al. [239] proposed to use the information from future frames to improve the detection of moving objects in the current frame. Although the currently utilised hand-craft features for moving object detection and tracking offer good results, new trends are toward employing more descriptive features. Nevertheless, the learning process can be used to target particular representations as opposed to a fixed collection of pre-defined traits [240]. CNN have recently been proposed for tracking moving objects, which effectively exploit category specific features for tracking objects even in complex scenarios like moving cameras [241; 242; 243]. CNNs have typically been employed for tracking in one of two ways: as a feature extractor in conjunction with a good classifier [244], or as a unified deep structure for object tracking [245]. Wang and Ying [244] proposed the use of Deep Learning Tracker (DLT) to learn genetic traits from supplemental natural images. Although

DLT being successful in scenarios with major temporal changes, the proposed method fails to learn these changes efficiently. Wang et al. [246] used an improved CNN architecture to train hierarchical features for model-free object tracking that can manage temporal variations and do online tracking quickly and in follow-up work, Wang et al. [247], employed a selection approach including two pre-defined convolutional layers to filter out noisy, irrelevant, or redundant features using CNN features retrieved from various layers. Zhai et al. [248] proposed a CNN tracker using a Bayesian classifier as a loss layer and updated the network parameters, which enables the detection of the moving object appearance over the time. Li et al. [249] published a comprehensive comparison of DNN-based tracking algorithms where it is highlighted that tracking CNNs are trained straightforwardly and effectively, resulting in good features for object tracking with the downside of requiring large annotated datasets.

Object tracking methods perform better than OMD algorithms in complex scenarios where both the cameras and objects moving freely. These approaches, however, substantially rely on the accurate initial selection of the moving object in the sequence's initial frames. Furthermore, tracking multiple objects is a complex problem to solve, especially when both the camera and objects are

moving freely. Although tracking is an important task in OMD, this PhD research programme does not focus on moving object tracking.

2.5.5.3 Real-time considerations

Although most methods for detecting moving objects work offline by analysing recorded video sequences, there is a need for real-time OMD, particularly in fields such as traffic monitoring, crowd tracking and ADAS [208]. Moreover, real-time applications require low-latency, which is normally preferred by parallelisable hardware such as GPUs and FPGAs and low-spec embedded CPUs [208; 250].

There exist a wide variety of objects which exhibit different behaviours, which makes tracking a complex process. Comaniciu et al. [251] proposed a method for real-time tracking of non-rigid objects captured from a moving camera, where the mean shift iterations are used to compute the target position in the current frame. Furthermore, the Bhattacharyya coefficient is used to express the dissimilarity between the target model (its colour distribution) and the target candidate. Comaniciu et al. [251] were able to demonstrate the tracker's ability to manage partial occlusions, considerable clutter, and target scale fluctuations in real time.

The object size impacts on the quality of object's tracking, and it becomes more difficult to track small objects or objects that become

progressively smaller when moving away from the centrer of the camera. Ponga & Bowden [252] proposed using probabilistic models for tracking small-area targets, which are common in outdoor visual surveillance scenes. The proposed model uses both appearance and motion models in the tasks of classification and tracking objects for which detailed information is not available. Moreover, the proposed method uses motion, shape cues, and colour information to distinguish between the different moving objects in the scene. The results show that the proposed method can track multiple people moving independently and maintain trajectories even when there are occlusions or background clutter.

The complexity of object tracking increases when it is required to perform multiple-object tracking because each object will describe random trajectories and might occlude or be occluded by other objects in the scene. Yang et al. [253] demonstrated a real-time method for tracking several objects in dynamic situations. The proposed method demonstrated that it is capable of coping with long-term and full occlusion without prior knowledge of the object's shape or motion. Extensive testing using video sequences in varied situations of indoor and outdoor environments with long-duration and complete occlusions in changing backgrounds demonstrated that the proposed method achieves good segment and track-

ing for images of 30×240 at 15 20 fps.

Selective tracking is also a challenge, especially when the real-time tracking of the target moving object is crucial in sport applications. Grabner et al. [254] proposed an online AdaBoost feature selection method for employing rapid computable features (such as Haar-like wavelets, orientation histograms, and local binary patterns) in real-time. Heinemann et al. [255] extended the work proposed by Grabner et al. [254] and explored tracking in a controlled environment and proposed a method for accurately recognising and tracking the ball in a RoboCup scenario[1]. To get a colourless representation of the ball, the authors used Haar features specified by the AdaBoost method [256] and particle filter for handling the ball tracking. It was demonstrated that, even in a congested environment, the proposed method can follow the ball in real-time (i.e. at 25 fps).

Tracking crowd movements is also a complex tracking task, because each pedestrian in the scene can describe a random trajectory and be occluded by other pedestrians. Shah et al. [257] propose an automated surveillance system that is suitable to be used in a wide range of real-world applications, including railway security and law enforcement. The proposed method has been utilised in a number of

[1]Available online, https://www.robocup.org/, last accessed: 12/04/2022

surveillance-related initiatives that have been supported by governments and private entities. The algorithm is capable of detecting and classifying targets, as well as tracking them across numerous cameras. It also creates a summary in terms of key frames and a textual description of trajectories for final analysis and reaction decision by a monitoring officer.

One of the most complex tracking challenges is associated with tracking unknown objects because the tracking algorithm must distinguish between visual artefacts (such as shadows or ghost objects) and real objects. Bibby et al. [258] proposed a probabilistic approach for robust real-time visual tracking of previously unknown objects captured by a moving camera. A bag of pixels representation is used to solve the tracking problem, which includes stiff frame registration, segmentation, and online appearance learning. The registration compensates for rigid motion, segmentation models any residual shape distortion, and the online appearance learning refines both object and background appearance models on a continuous basis.

Night objects tracking is also a challenge for real-time object tracking applications because it is difficult to distinguish between the target moving objects and shadows generated by artificial illumination. Huang2008 et al. [259] proposed a real-time object

recognition technique for nighttime vision surveillance through contrast analysis. The contrast in local changes over time is employed to detect moving objects in the scene, and the false positives are reduced by combining motion prediction and spatial closest neighbour data association.

Object adaptability is a characteristic of a good real-time tracking algorithm. Li et al. [260] proposed the use of mean shift technique-based solution for global target tracking combined with a background weighted histogram and a colour weighted histogram to represent the model and the candidate. The results show the proposed method may adaptively achieve precise object size with low computational complexity, including camera motion, camera vibration, camera zoom and focus, high-speed moving object tracking, partial occlusions, target scale variations, etc.

Intrinsic characteristics of objects may also have a positive impact on the performance and speed of the algorithm. Danelljan et al. [216] proposed a method that performs real-time object tracking using colour features. The findings show that colour features provide greater visual tracking performance and offer a low-dimensional adaptive colour attribute variation for real-time aspects. The authors also demonstrated that the proposed method is comparable to state-of-the-art tracking algorithms, with a surprising speed of

up to 100 fps.

Navigational applications are amongst the biggest challenges for real-time object tracking algorithms that must be capable of tracking objects of interest in a very short period of time to adjust the trajectory of the moving vehicle. Agarwal et al. [261] proposed a real-time multiple object tracking using a Region Based Convolutional Neural Network (RBCNN) for object detection and a regression network for general object tracking specially design for targetting ADAS applications. Minaeian et al. [262] proposed the use of local motions for detecting moving objects and extract tracking keypoints for images captured using Unmanned Aerial Vehicles (UAVs). Their efforts yielded positive outcomes in real-world applications. Overall, it is possible to infer that reliable OMD algorithms are required in a wide range of applications and their quality will dictate the quality of advanced OMD applications (such as trajectory classification and object tracking). Although advanced OMD applications were covered in this section, this PhD research programme does not focus on such applications.

2.6 Hardware implementations

The neuromorphic computing concept was introduced by Carver Mead [263] in 1990 and was described as the use of Very Large

Scale of Integration (VLSI) equipped with analogue components for emulating biological neural systems. Neuromorphic architectures deliver flexibility for describing highly parallel architectures, require low-power and are typically interconnected to CPU via high-speed interfaces (such as PCIe or intra-chip high-speed bus) [264].

Neuromorphic architectures can perform parallel calculations faster, with higher power efficiency and a smaller footprint than the traditional 32/64-bit standard von Neumann architectures [264]. Furthermore, the emerging AI/ML methods require more flexible hardware architectures to accommodate the unique requirements of such methods[264]. Emerging neuromorphic architectures deliver higher densities of transistors per unit of space, higher intra-chip communication speed between the customisable fabric, real-time capabilities, and other application-specific processing units (e.g. CPUs and GPUs) [28]. Neuromorphic implementations can be split into digital, analogue, and hybrid (both digital and analogue) platforms [264]. This PhD research programme aimed to develop a digital neuromorphic solution to accommodate and accelerate the customised SNN using digital systems [264]. Neural Network Accelerators (NNA) are networks of neuromorphic chips optimised to handle complex neural network workloads that require reconfigurable com-

putational resources and a high degree of parallelism. The comparison between different types of NNA still remains a challenge, and there is no simple way to compare the performance between ANNs with SNNs nor between different types of Neural Networks (i.e. different types of ANNs or SNNs). This PhD research programme focused on SNNs, and therefore, this section only reviews NNA that were specially designed for accelerating SSNs [15].

2.6.1 General propose Neural Network accelerators

SpiNNaker [265] is a network composed of custom CPUs with an architecture optimised for running customised SNNs based on LIF neurons. BrainScaleS [266] is another powerful NNA composed of interconnected wafers composed of high input count analogue neural network cores targeting the rigorous emulation of brain-scale NNs. Neurogrid [267] is an analogue NNA for emulating the structure of biological nervous systems in real-time. TrueNorth [268] is a digital and low-power neuromorphic chip for simulating complex neuronal networks. Loihi [269] neuromorphic chip is a state-of-the-art brought by Intel specially designed for enabling on-chip learning compatible with several learning rules, complex neuron and synapses models targetting complex SNNs

Table 2.2 summarises the characteristics (implementation, sampling time, of each one of the NNA).

Table 2.2 Large-scale Neural Networks accelerators characteristics. Adapted from [15]

Processor	BrainScaleS [266]	Neurogrid [267]	TrueNorth [268]	SpiNNaker [265]	Loihi [269]
Implementation type	analogue	analogue	digital	digital	digital
Sampling Type	discrete	continuous	discrete	discrete	discrete
Neuron Update	continuous	continuous	Time MUX	Time MUX	Time MUX
Synapse Resolution	4b	13b shared	1b	Variable	1 to 64b
Bio-Mimicry	Not Configurable	Not Configurable	Limited to LIF	Configurable	Configurable
On-Chip Learning	STDP only	No	No	Yes	Yes
Network on Chip	Hierarchica	Tree Multicast	2D Mesh Unicast	2D Mesh Multicast	2D Mesh Unicast
Neurons per Core	8 ∼512	65e3	256	∼1e3	max. 1024
Synapses per Core	∼130k	100e6	65k x 1b	∼1e6	16000 x 64b
Cores per Chip	352 (wafer scale)	1	4096	16	128
Chip Area (mm2)	50 (single core)	168	430	102	60
Technology (nm)	180	180	28	130	14 (FinFET)
Energy/SOP (pJ)	174	941	27	27e3	105.3

2.6.2 Neuromorphic and heterogeneous devices

Lichsteiner *et al.* [270] introduced the concept of Address Event Representation (AER) silicon retina chip capable of generating events proportional to the log intensity changes. Farian *et al.* [271] proposed an in-pixel colour processing approach inspired by the retinal colour opponency using the same AER concept. Brandli *et al.* [272] proposed a new version of the AER camera reported by Lichsteiner [270] called the Dynamic Active Pixel Vision Sensor (DAVIS) which exploits the efficiency of the AER protocol and introduces a new synchronous global shutter frame concurrently.

Kasabov et al. [273] proposed the deSNN that combines the

use of Spike Driven Synaptic Plasticity (an unsupervised learning method for learning spatio-temporal representations) with rank-order learning (supervised learning for building rank-order models). The deSNN [273] was tested on data collected by an AER silicon retina chip [270] (which generates events in response to changes in light intensity) for recognising moving objects. Although the deSNN was able to recognise moving objects, the deSNN was designed to work specifically with AER cameras. More recently, Jiang *et al.* [274] proposed a SNN based on the Hough Transform to detect a target object with an asynchronous event stream fed by an AER cameras. The proposed algorithm [274] was able to process up to 40.74 frames per second on an Intel i7-4770 processor, accelerated by an Nvidia Geforce GTX 645. Nevertheless, the authors do not explain if the algorithm would work with regular Commercial-Off-The-Shelf (COTS) cameras.

Oudjail & Martinet [275] proposes an approach to analyse a moving pattern motion using SNN from feed captured using Dynamic Voltage Scaling (DVS) AER camera. A synthetic dataset containing three different moving patterns in four directions was used to train and test the custom SNN designed by Oudjail & Martinet [275]. The proposed SNN [275] is composed of LIF neurons whose weights were trained using STDP unsupervised learning [8] and the

algorithm was implemented in the Brian simulator [55]. The authors concluded that the number of output neurons (between 2 and 6 in the output layer) is insufficient to detect the movement's direction accurately. The results of Oudjail & Martinet [275] work raise the following questions: 1) Was the number of neurons in the input layer sufficient? Because each input neuron received the currents of 150 light intensity values encoded in currents, which would most likely trigger the middle layer neurons to over-spike, forcing the post-synaptic weight to increase until reaching the maximum allowed value; 2) Was the number of neurons on the output layer sufficient? Each output neuron received the synaptic connections from 25 of the previous layer neurons, again the 25 neurons which would most likely trigger the middle layer neurons to over-spike pushing the post-synaptic weight to increase until reaching the maximum allowed value during the training phase; and 3) Supervised learning ReSuMe (see Chapter 3 for further details) could have been used to train the output layer neurons because the desired output pattern was well known and could have been used to teach the output layer to recognise the desired patterns.

Jiang et al. [274] proposed a SNN based on the Hough Transform to detect a target captured using a DVS AER camera. The authors concluded that the speed performance of the SNN on the

GPU was better than on the CPU [274]. Analysis of the results of Jiang's work [274] leads to the conclusion that the work is incomplete because 1) little is stated about the dataset; 2) it is not clear how the dataset was preprocessed; 3) no metrics were used to assess the accuracy of the tracking system; and 4) the comparison between the GPU and CPU speed performance is insufficiently rigorous because a simplified and synchronous update rule was only used on the GPU.

Kuriyama et al. [276] proposed to accelerate the cerebellar scaffold model [277] using heterogeneous computing based on GPUs. They simulated synaptic plasticity mechanisms at parallel fibre - Purkinje cell synapses and emulated the gain adaptation of optokinetic response. The results show that the use of GPUs enables the processing of 2s of simulation in just 750ms, which is about 100 times faster than previous simulations running on CPUs. Although the Kuriyama et al. [276] model achieves an impressive acceleration of 100 times faster than the CPU, the acceleration was obtained using 4 GPUs which is not cost-effective (each NVIDIA Tesla V100 GPU used in this project cost thousands of dollars). It will not be easily scalable (the majority of modern servers will only have up to 4 PCIe slots to install GPUs and other devices). Moreover, Kuriyama did not include details about the test conditions of the

CPU implementation nor the CPU specifications, which makes it difficult to assess if the comparison between the GPU and CPU was fair. The emulated circuit was not a retinal like circuit, which is the focus of this research project.

She et al. [278] proposed a heterogeneous SNN (H-SNN) suitable for learning complex spatiotemporal patterns when using unsupervised learning STDP. The authors [278] demonstrated analytically that the H-SNN exhibits long and short memory capabilities. It was established, in simulation, that the H-SNN is capable of classifying the object type and motion dynamics. A GPU was used to accelerate the SNN [278], but little details are provided about the level of acceleration achieved. The results reported in [278] focus on the accuracy of the H-SNN when compared to other classical CNNs, and it is not clear how accurate the H-SNN is predicting the motion dynamics. Parameshawara et al. [279] proposed the SpikeMS, a deep SNN encoder-decoder for motion segmentation of AER frames captured using DVS AER cameras. A novel spatial-temporal loss formulation based on spike counts and classification labels is used on the SpikeMS. The SpikeMS predicts using time windows with a duration of 10 ms. Although the SpikeMS can predict the object type with high accuracy using a 10 ms time window, it is focused on DVS AER cameras, and it is not clear how it would perform in

COTS cameras.

2.6.3 FPGA implementations

FPGAs are specialised devices that can be reprogrammable after manufacture and offer high flexibility, high degree of parallelism, high-performance and low-power platforms [280]. FPGA offers the desirable flexibility for accelerating SNNs which are characterised for being massively parallel [281]. FPGAs are also known for being programmed using low-level HDLs such as Very High Speed Integrated Circuit Hardware Description Language (VHDL), and Verilog [282]. Nowadays, FPGA development has been simplified with the introduction of High Level Synthesis (HLS) tools, which allow FPGA developers to implement their applications using high-level programming languages such as C and OpenCL [282].

Mishra et al. [283] identified in their survey that many of SNN usually have about $10^4 \sim 10^8$ neurons and $10^{10} \sim 10^{14}$ synapses and that high-performance neural hardware is essential for practical application. Li et al. [284] proposed the implementation of visual cortex neurons on FPGAs. The implemented visual cortex neurons exhibited the same dynamics as those recorded from real neurons using multi-electrode arrays. Li et al.[285] implemented

256 fully connected neurons, and their performance was assessed by storing four patterns and applying similar patterns containing errors. The implemented system was capable of operating using a 100 MHz clock, which enables the acceleration of the system 40 times above its real-time operation [285]. Cassidy et al. [286] proposed the use of FPGAs to accommodate spiking neurons and unsupervised STDPSTDP learning structures. In this work, Cassidy et al. [286] demonstrated that digital neuron abstraction is preferable to more realistic analogue neurons; they also emulated the massive parallelism connectivity and high neuron density as observed in nature; the neuron states were also multiplexed to take advantage of clock frequencies and dense Static Random Access Memorys (SRAMs).

Moeys et al. [287] implemented object motion cells in FPGAs that takes about 22 clock cycles at 50 MHz to detect motion and also reported that the FPGA is at least 100 times less than an Intel Next Unit of Computing (NUC) to compute motion. Furthermore, the work in [287] shows that the FPGA implementation has lower latency when compared with the same implementation running on Intel NUC at 1.30 GHz and an Intel I7-4770k at 3.50Ghz.

Chen et al. [288] described a Central Pattern Generator (CPG) composed of two reciprocally inhibitory neurons. To reduce the FPGA resources usages, Chen et al. [288] has optimised the CPG

to avoid using multipliers (FPGAs have a low quantity of multiplier blocks), and the non-linear parts of the Komendantov-Kononenko neuron model [48] were removed. Cheung et al. [289] proposed the NeuroFlow, a scalable SNN simulator suitable to be implemented on FPGA clusters. It was possible to simulate about 600,000 neurons and to get a real-time performance for up to 400,000 neurons simulated using NeuroFlow on 6 FPGAs [289]. Podobas & Matsuoka [282] proposed the use of OpenCL, an HLS tool, to increase productivity by facilitating the SNN design (provide a higher level of hardware abstraction) on FPGAs. Two different neuron models, their axons and synapses, were designed using OpenCL and Podobas & Matsuoka [282] claim a speed performance of up to 2.25 GSpikes/second. Sakellariou et al. [290] suggested a spiking accelerator based on FPGAs to enable users to develop SNNs targetting ML applications and promise an acceleration of up to eight hundred times for inference and up to five hundred times for training compared to Software SNN simulations.

The works reviewed in this section demonstrated that FPGAs offer flexibility, high efficiency, low-power, and high degree of parallelism, making FPGAs suitable devices for implementing brain-like circuits. Furthermore, FPGAs enable the design of complex biologically plausible neuron models and massively parallel SNNs capable

of generating complex biological like patterns. Although FPGA devices being normally programmed using complex HDL tools, HLS tools such as OpenCL can be used to increase the productivity of SNNs design process by providing hardware abstraction which reduces the implementation complexity.

2.7 Revised Literature

The works reviewed in section 2.1 highlight that retinal cells are organised in multi-hierarchical circuits. Unlike other retinal cells, the retinal ganglion cells trigger spike events encoded and forwarded to the visual cortex via the optical nerve. OMS-GC natural circuits include both nor-spiking with spiking cells, and therefore efficient OMS-GC computational models will most likely combine non-spiking BS models (e.g. BS mathematical theory models) with spiking neuron models (i.e. customised SNNs). The works revised in sections 2.2 and 2.3 show that SNNs are biologically plausible, capable of producing realistic patterns, are hardware-friendly and therefore suitable to be accelerated using dedicated hardware. SNN have a massively parallel component because they combine incorporate spiking neurons connected via numerous synapses, with differing synaptic models. The SNN parallelism introduces speed perfor-

mance limitations if implemented on traditional CPUs based on the von Neumann architecture have a massively parallel component because they combine incorporate spiking neurons connected via numerous synapses, with differing synaptic models. This SNN parallelism introduces speed performance limitations if implemented on traditional CPUs based on the von Neumann architecture (see Chapter 4 for further details). Several OMD methods were also reviewed in section 2.5. Object Motion Detection includes 4 phases, namely, Background subtraction, noise reduction, threshold selection and moving objects detection (see Figure 2.8). Three main classes of BS were identified, namely, mathematically based theories, ML/DNN, signal processing models. Mathematical based models were selected for this PhD research programme because these methods require lower computational requirements when compared with ML and signal processing models. Although BS mathematical based models methods show a good balance between robustness and processing speed, these modules can be improved using SNNs to 1) produce bio-realistic responses for neuro-engineering applications (such as eye prosthesis), 2) enhance BS methods to filter high-frequency parasitic light variations, and 3) reduce power by reducing the dynamic power consumption (using lower speed clocks). The enhancement of the GSOC BS algorithm using the SNNs is

reported in Chapters 4 and 5. Although the OMD steps 2) noise reduction and 3) threshold selection are relevant, this PhD research programme will also explore the use of SNNs to improve the step 4) Moving objects detection (see Chapter 3) by combining receptive fields (acting as filters to reduce noise) and synapses with different propagation delays (short term memory) and lateral inhibition (improving the classification).

Neuromorphic computing using VLSIs or FPGAs have been used to accelerate for accelerating SNNs and other brain-like functions. Modern System-on-Chip solutions make use of heterogeneous computing for accelerating SNNs using CPUs with GPUs, VLSIs and FPGAs. The revised literature shows that FPGAs deliver more flexibility but with higher computational complexity because the hardware developers are generally required to have a deep understanding of FPGA devices (see Chapter 5.2.2 and 5.2.3 for further details). GPUs are more straightforward to program (developers are abstracted of the GPU hardware architecture) than FPGAs and are suitable for accelerating classical ML/AI algorithms and delivering FPGA comparable speed performances. Nevertheless, the GPU architecture has been designed to accelerate operations with 2D matrices. VLSIs are normally the most power-efficient devices and perhaps should be faster than FPGAs and/or GPUs, but

VLSIs have a well-defined architecture. Moreover, state-of-the-art DNN algorithms require flexibility for changing the number of neurons, synapses, and weights and VLSIs do not offer this flexibility. Therefore, FPGAs were selected to be used in this PhD research programme.

This PhD research programme investigated the i) use of customised SNNs for motion detection using either MHSNN in Chapter 3; (ii) enhance existing mathematical theory BS algorithms using a customised SNN targetting real-time applications in Chapter 4 and iii) accelerate the customised SNN using an FPGA in Chapter 5.

Chapter 3

Detection of horizontal and vertical movements using Spiking Neural Networks

The MHSNN architecture for detecting horizontal and vertical movements using a custom dataset is proposed in this chapter. Furthermore, the dataset is composed of semisynthetic image sequences of black cylinders performing rightwards, leftwards, upwards, and downwards movements, which were generated in laboratory settings. The MHSNN architecture was designed to reflect the connectivity, behaviour, and the number of layers found in the majority of vertebrate's retinas, including humans. The architecture was trained using 2303 images and tested using 816 images. Simulation results revealed that each cell model is sensitive to vertical and horizontal

movements, with a detection error of 6.75%.

3.1 Introduction

Vertebrate retinas have the ability to sense the direction of moving objects. [7]. The DSGC detect motion direction by spiking strongly when an object moves in the preferred direction and sparsely when the identical object moves in the opposite direction [18; 291; 292]. These cells are composed of bistratified dendrites that laminate in both the On and Off of the retina's Inner Plexiform Layer (IPL). Dendrites of glutamatergic bipolar cells supply primarily excitatory stimuli, while starburst amacrine cells (SAC), supply primarily inhibitory stimuli (see Figure 3.1) [292].

The MHSNN , proposed in this chapter, was designed for mimicking DSGC basic functionalities (i.e. detection of horizontal and vertical movements) [293]. Similar to the DSGC, the MHSNN is also composed of 4 layers, each layer designed to deliver functionalities of specific vertebrate retinal cell (i.e. photoreceivers, bipolar, SAC and DSGC cells). Furthermore, MHSNN combines the use of synapses with different propagation delays for generating residual synaptic memories, and also combines inhibitory with excitatory synapses for the generation of patterns associated with the direction

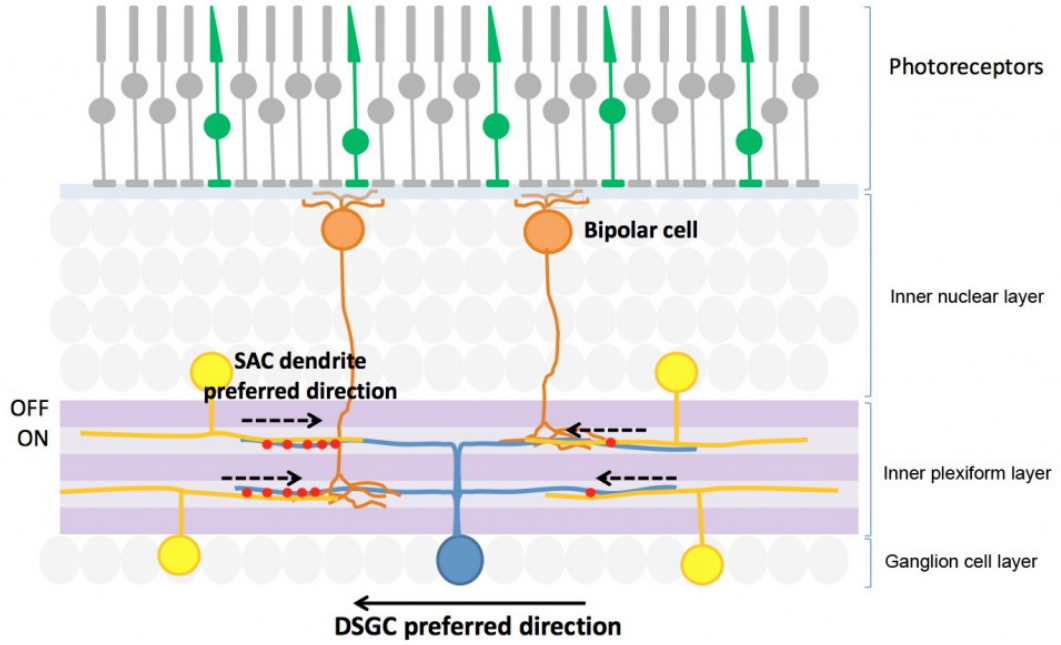

Figure 3.1: Schematic of the DSGC circuit. The figure shows DSGC (in blue), SAC dendrites (in yellow), bipolar cells (in orange) and photoreceptors (green). Inhibitory synapses (in red dots) are formed on the DSGC by the SAC dendrites (dashed arrows), which have a preferred movement in the opposite direction to the DSGC (solid arrow) [7].

taken by moving objects.

Custom object direction detection (CODD) algorithm (described in Section 3.3.4 was designed and implemented to establish the comparison between the accuracy and computational speeds against the MHSNN algorithm. The CODD algorithm is based on the BS algorithms available through the OpenCV library. The OpenCV library is maintained by a large Open Source community (including well-known corporations like Intel, NVIDIA, Microsoft, and Google) and

it is one of the most popular and reliable computer vision libraries.

The chapter structure is as follows: the proposed MHSNN is described in Section 3.2, the simulation methodology is described in section 3.3, the simulation results are presented in Section 3.4 and analysis and future work are discussed in Section 3.5.

3.2 Proposed architecture

The MHSNN (see Figure 3.2) is a four-layered architecture proposed for detecting vertical and horizontal movements. The input layer converts the normalised graded pixel values (0.0 up to 1.0) to spike events (where values above 0.85[1] are considered a spike event); Layer 1 detects local edges; Layer 2 extracts movement direction features, Layer 3 extracts movement features, and Layer 4 detects types of movement.

3.2.1 Input Layer: Binarisation via conversion from pixel grade values to spike events

The input layer converts graded pixel values to spike events. Values equal to or above 0.85 are considered spike events, similar to the functionality of rods [21]. A 1:1 connectivity is used between each pixel and the neurons in the input layer.

[1]The value 0.85 was obtained empirically.

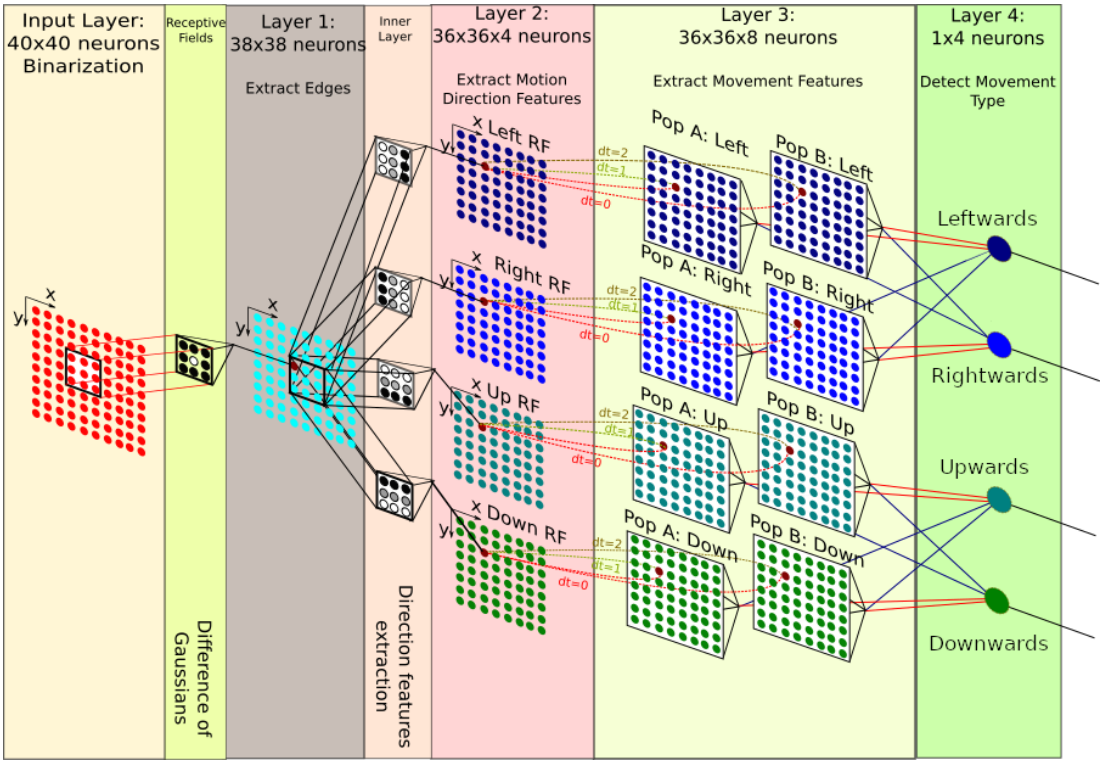

Figure 3.2: MHSNN with (i) 40×40 image input followed by the four processing Layers. Layer 1: Edge detection Layer, Layer 2: Direction features extraction, Layer 3: Movement extraction features and Layer 4: Direction-sensitive ganglion cells.

Figure 3.3 shows an example where three sequential image frames (from a synthetic dataset with objects performing horizontal and vertical movements) of 40×40 pixels are presented to the Input Layer, then processed by the neurons in Layers 1 to 4. The number of required synapses and neurons is given by equations 3.1 and 3.2.

$$N_N = (L-2).(W-2) + 3.M_f.(L-4).(W-4) + M_f \tag{3.1}$$

$$N_S = F_L.F_W.\left[(L-2).(W-2) + 3.M_f.(L-4).(W-4)\right] + 4.M_f.(L-4).(W-4) \quad (3.2)$$

where the N_N is the number of neurons, N_S is the number of synapses, L is the length of the image in pixels, W is the width of the image in pixels, M_f is the number of movement features, F_L is the filter length and F_W is the filter width.

The decision to use small-sized images was taken based on the number of synapses and neurons that were required to implement the MHSNN. A total of 17000 neurons and, 173700 synapses are required to build the network for processing images of 40×40 pixels. Overall, it takes 6 minutes and 30 seconds to build the MHSNN architecture, 6 hours and 37 minutes for training and 7 seconds to run the simulation when using a workstation equipped with an 8-core Intel Xeon E5 2640 CPU @ 2.60GHz, 96 GB of DDR4, and 1 TB of disk space. Although 6 minutes and 30 seconds for building the network and 7 seconds for running the simulation seems to be an acceptable timing, the results shown in the Tables 3.3, 3.4, 3.1 and 3.2 show that 7 seconds is about 887% slower than the CODD algorithm.

Figure 3.3 shows a black cylinder object moving (small displacements) rightwards. In the example, each image frame is exposed to

the input layer for a simulation period of 1ms. The results are represented with vertical bars (i.e. spike events) in the output synapse of the DSGC rightwars cell, which is represented with an R.

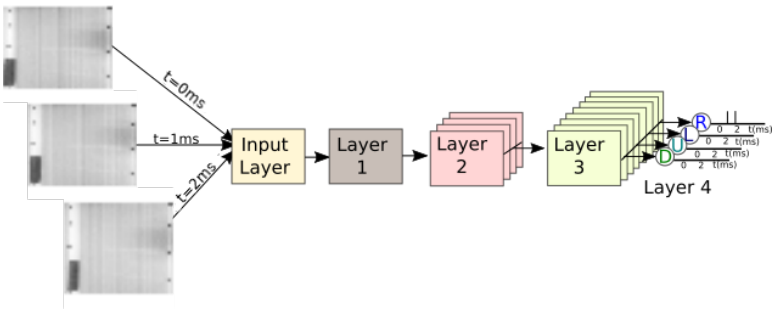

Figure 3.3: Three image frames being processed by the proposed architecture. The images are exposed to each Layer in sequence (Layer 1, 2, and 3), and finally, the movement is detected in Layer 4 by rightwards (R), leftwards (L), upwards (U) and downwards (D) GC.

3.2.2 Layer 1: Edge detection

The aim of Layer 1 is to detect edges. 40×40 pixels images are exposed to Layer 1 for a period of 1 simulation time step. The neurons in Layer 1 receive spike events from a 3×3 patch where the central neuron is connected via an excitatory synapse (weight greater than 0), and the eight neighbouring neurons are connected via inhibitory synapses (weight lower than 0), originating a Receptive Field (RF). The RF in the visual system comprises a 2D region in a specific visual space and may have different sizes and shapes. The latter is represented in Figure 3.2 by windows composed of 8

black circumferences (acting as inhibitory synapses) and a white central circumference (acting as an excitatory synapse). The RFs have a stride of 1 for retaining the spatial information required for performing the edge detection of a RF weights' distribution given by the DoG function [294]. The DoG function, commonly used to perform edge detection[294], in Layer 1 is used to perform the edge detection and is governed by equation 3.3.

$$DoG(x,y) = \frac{1}{2\pi\sigma_s^2} e^{-\frac{x^2+y^2}{2\sigma_s^2}} - \frac{1}{2\pi\sigma_c^2} e^{-\frac{x^2+y^2}{2\sigma_c^2}} \tag{3.3}$$

$$\frac{\sigma_c}{\sigma_s} = 1.6 \tag{3.4}$$

Because of the border conditions, the number of neurons per pixel is reduced by two columns and two rows (i.e., the required number of neurons is reduced by 38×38). Zero or negative values are produced by the DoG filter when a given patch is composed of similar pixel intensities, preventing neurons from generating spike events; Positive values are produced when the neighbouring pixels and the central pixels have a considerable variation, triggering neurons to generate spike events.

3.2.3 Layer 2: Horizontal and vertical features extraction

The aim of Layer 2 is to extract horizontal and vertical features.

Layer 2 is composed of 36 rows× 36 columns×4 groups of neurons (36×36×4). Each neuron is connected via a 3×3 RF, and the connection between neurons is performed via inhibitory and excitatory synapses. The excitatory synapses between the neurons of Layer 1 and Layer 2 vary according to the type of filter used. The goal of each filter type is to generate unique spike patterns that are required for detecting the movement type while retaining spatial information. The filters in the inner Layer are represented in Figure 3.2 with nine blue circles overlapping nine neurons in Layer 1.

The neurons in Layer 2 are used to extract features related to each type of movement (rightwards, leftwards, upwards and downwards). Again, 3×3 windows are used to produce features maps that capture the required details for a specific movement and distinct pattern from the homologous movement.

$$L_f\left(x,y\right) = \sum_{i=0}^{x}\sum_{j=0}^{y}\frac{x}{2} - i \tag{3.5}$$

$$R_g\left(x,y\right) = \sum_{i=0}^{x}\sum_{j=0}^{y}\frac{-x}{2} + i \tag{3.6}$$

$$U_p(x,y) = \sum_{j=0}^{y} \sum_{i=0}^{x} \frac{-y}{2} + j \tag{3.7}$$

$$D_w(x,y) = \sum_{j=0}^{y} \sum_{i=0}^{x} \frac{y}{2} - j \tag{3.8}$$

where the parameters L_f, R_g, U_p and D_w represent rightwards, leftwards, upwards, and downwards filter weight distribution. The number of neurons per patch of layer neurons is reduced by two columns and two rows (i.e.. The required number of neurons is 36×36) per filter type because of the border conditions. Like the DoG filters, neurons will trigger spike events when the current values generated by the patch of neurons in Layer 1 is positive and greater or equal to the threshold.

3.2.4 Layer 3: Extraction of movement features

The aim of Layer 3 is to extract horizontal and vertical movement features. Layer 3 neurons are connected in a one-to-one (1:1) configuration to the Layer 2 neurons. There are two neuron populations, Population A and Population B.

The Population A neurons are wired via two synapses, an excitatory synapse with no delay, and an inhibitory synapse with a 1ms delay. The use of synapses with different delays facilitates movement feature extraction [295]. In addition, the delay produced by

synapses with varying times of propagation creates a short-term memory. E.g. given a pre-neuron that spiked at the timestep t and did not spike at timestep $t-1$ causes a spike event in the post-neuron because the difference between spike events is positive and above the threshold.

Population B differs from Population A because the inhibitory synapses are delayed by two simulation time steps instead of one. The two populations are used for improving the accuracy of the movement features extraction by creating a bio-inspired buffer of 3 spike patterns. Each group of neurons comprises groups of 36×36 neurons, and in the overall configuration, there are eight groups of neurons (four per population). The neurons in layer three can sense movement because they are interconnected using different propagation delays. Such delays create a local residual memory for holding past neurons' events and comparing them with current events (the simulation time-step was set to 1ms). In addition, the populations are used to improve the robustness of the motion detection because fast movements are detected by Population A, slower signs by Population B and average speed movements are seen by both populations of neurons. Therefore, a movement feature is extracted if a spike in the current frame was not detected in the previous frame (population A with a propagation delay of one) or two frames before

(population B with a propagation delay of 2).

Therefore, it is possible to generate different spiking patterns to detect differences between spike trains. The neurons in Layer 3 will spike if changes are detected.

3.2.5 Layer 4: Detection of movement type

The aim of Layer 4 is to detect horizontal and vertical movements' directions (i.e. rightwards, leftwards, downwards, and upwards) based on the movement features extracted in Layer 3. The horizontal and vertical movements were inspired by the basic functionalities exhibited by DSGC [7; 18]. Each neuron in Layer 4 receives connections from all the neurons of population A and B, of its specific type of movement (*e.g.* right movement cell is connected via excitatory cells to the left population's A and B) and inhibitory synapses from its type of movement (*e.g.* right movement cell is connected via inhibitory cells to the right population's A and B). The reason for having these connections is that the left cell must not spike when the right cell is spiking, or vice-versa. It is possible in certain circumstances to have different types of cells spiking at the same time (*e.g.* a simultaneous rightwards and upwards). However, a particular cell cannot spike at the same time as its pairing cell (*e.g.* the right cell cannot spike at the same time as the left cell because

that would mean that a given object was moving rightwards and leftwards at the same time). In Figure 3.2, the inhibitory synapses are represented with dashed red lines, and the inhibitory synapses with blue lines. The brown and yellow dashed boxes indicate that all the neurons are connected to the movement-sensitive cells.

This layer contains four types of neurons, two of which respond to horizontal movements and the other two to vertical movements. These neurons receive connections from the neurons of both populations, A and B, of the same type and its pairing (*e.g.* right/left). The connections from the same kinds of movements are excitatory, while the connections from the paring type are inhibitory.

The weights of these connections were trained using the ReSuMe algorithm proposed in [8].

ReSuMe [8], a supervised learning algorithm, was used to train the Layer 3 neurons' response. In ReSuMe, teacher signals are used to produce the desired spike pattern in response to a stimulus [8].

Figure 3.4 shows the teacher signal (desired spike pattern) n_{teach} being presented to a neuron n_{post} for delivering a spike pattern by adjusting the synaptic weight w between the pre-neuron n_{pre} and the post-neuron n_{post}. The learning occurs with the modification of the weights.

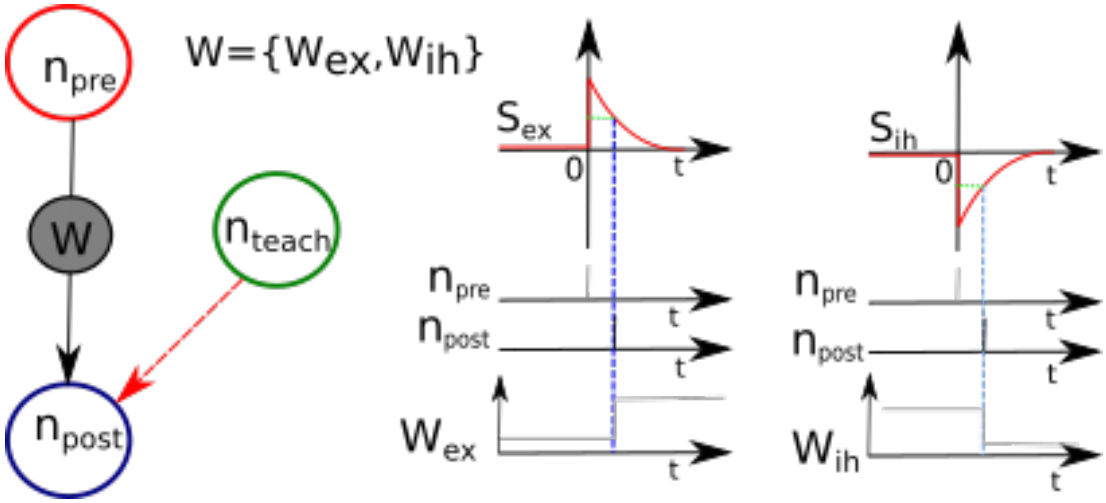

Figure 3.4: ReSuMe learning: (left) Remote supervision. (right) Learning windows [8].

The ReSuMe [8] equations are as follows:

$$[h]W_{ex}(s_{ex}) = \begin{cases} A_{ex}e^{\left(\frac{-s_{ex}}{\tau_{ex}}\right)}, & \text{if } s_{ex} > 0, \\ 0, & \text{if } s_{ex} \leq 0, \end{cases} \tag{3.9}$$

$$W_{ih}(s_{ih}) = \begin{cases} A_{ih}e^{\left(\frac{-s_{ih}}{\tau_{ih}}\right)}, & \text{if } s_{ih} > 0, \\ 0, & \text{if } s_{ih} \leq 0, \end{cases} \tag{3.10}$$

where A_{ex}, A_{ih}, τ_{ex} and τ_{ih} are constants. A_{ex} and A_{ih} are positive in excitatory synapses and negative in inhibitory synapses. In both cases, τ_{ex} and τ_{ih} are positive time constants [8].

Layer 4 neurons were trained using teacher signals. The teacher signal is generated using one teacher neuron (n_{teach}) connected to a Layer 4 neuron. To generate a spike event, a high current is applied

to n_{teach} which makes the Layer 4 neuron generate a spike event. This interactive process enables the weights of synapses coming from Layer 3 neurons to increase over time. The training was initially done on the horizontally sensitive cells and then on the vertically sensitive cells.

In Figure 3.5 the two teacher signals used to train the synaptic weights of the horizontal cells during the simulation time window [2290, 2315]ms are shown. The period [2290, 2315]ms refers to a time window where a black cylinder object is moving leftwards during the period [2290, 2303]ms rightwards during the period [2304, 2315]ms.

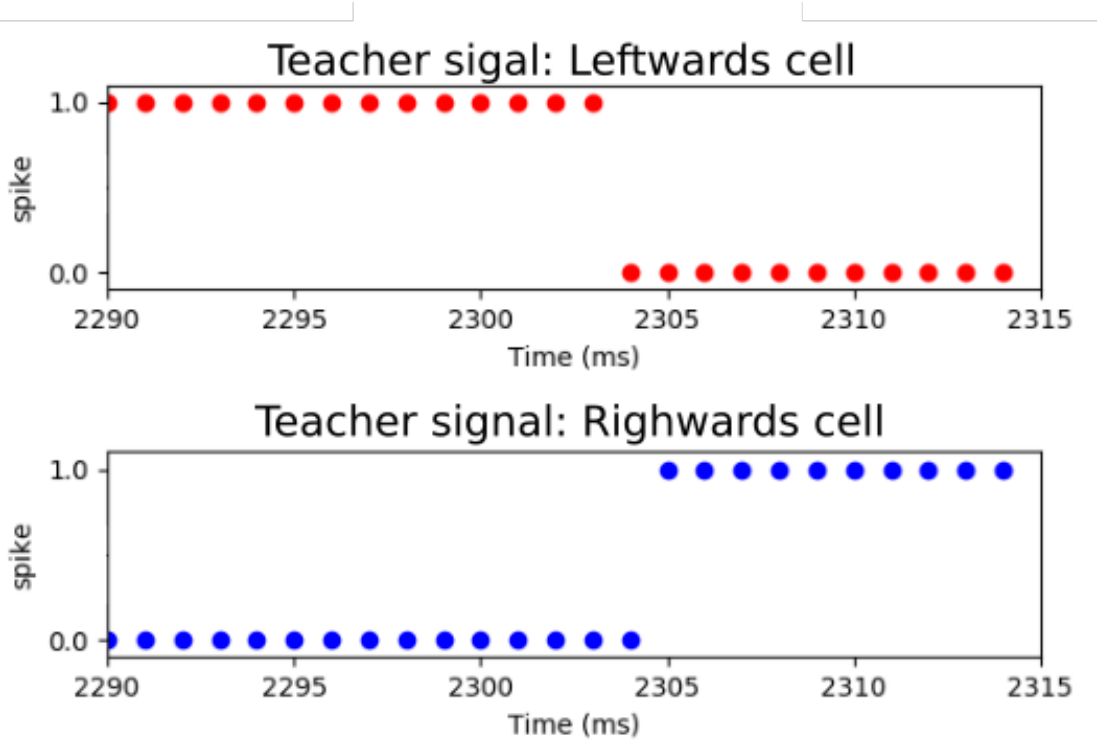

Figure 3.5: Two teacher signals used to train the horizontal sensitive cells during the simulation time window [2290, 2315]ms.

3.3 Implementation

This section covers the details of how the dataset was generated and preprocessed, and the methodology that was followed for evaluating the performance of the proposed MHSNN for detecting horizontal and vertical movements.

3.3.1 Dataset

A simple dataset containing 100 repetitions of a black cylindrical object moving rightwards. Each repetition comprises approximately 30 images (the number of images is proportional to the object speed). A total of 3120 images were generated from one hundred trials. The dataset was augmented by performing rotations of $\frac{\pi}{2}$ rad (downward movements), π rad (rightward movements), and $\frac{2\pi}{3}$ rad (upward movements). In addition, all the images were annotated with the type of movement being described. The original size of each figure is 640×480 pixels.

3.3.2 Image pre-processing

Natural images extracted from batches of image sequences require preprocessing to reduce the dimension of the images and lower the number of neurons per layer. Therefore, the image is first converted to greyscale and the number of channels is reduced from 3 (red, green, and blue) to one (greyscale) by applying equation 3.11 [296].

$$Grey = (0.299 \times R) + (0.587 \times G) + (0.114 \times B) \quad (3.11)$$

The second preprocessing step was to compute the PCA whiten-

ing to reduce the amount of redundant input. Through the application of PCA whitening, we minimise the degree of correlation between adjacent pixels or feature values, which might be highly correlated [297].

The PCA and whitening were done using the Python 3.8 Sklearn library[1]. The third and final step was resizing the image to reduce the number of required neurons per layer using the OpenCV library built-in functions for Python 3.8. The image frames were resized to 40×40 pixels while keeping the original aspect ratio.

For example, figure 3.6 shows a sequence of 4 images extracted from one of the trials, where the black cylindrical object is moving rightwards.

[1] Available online, https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html, last accessed: 22/06/2022

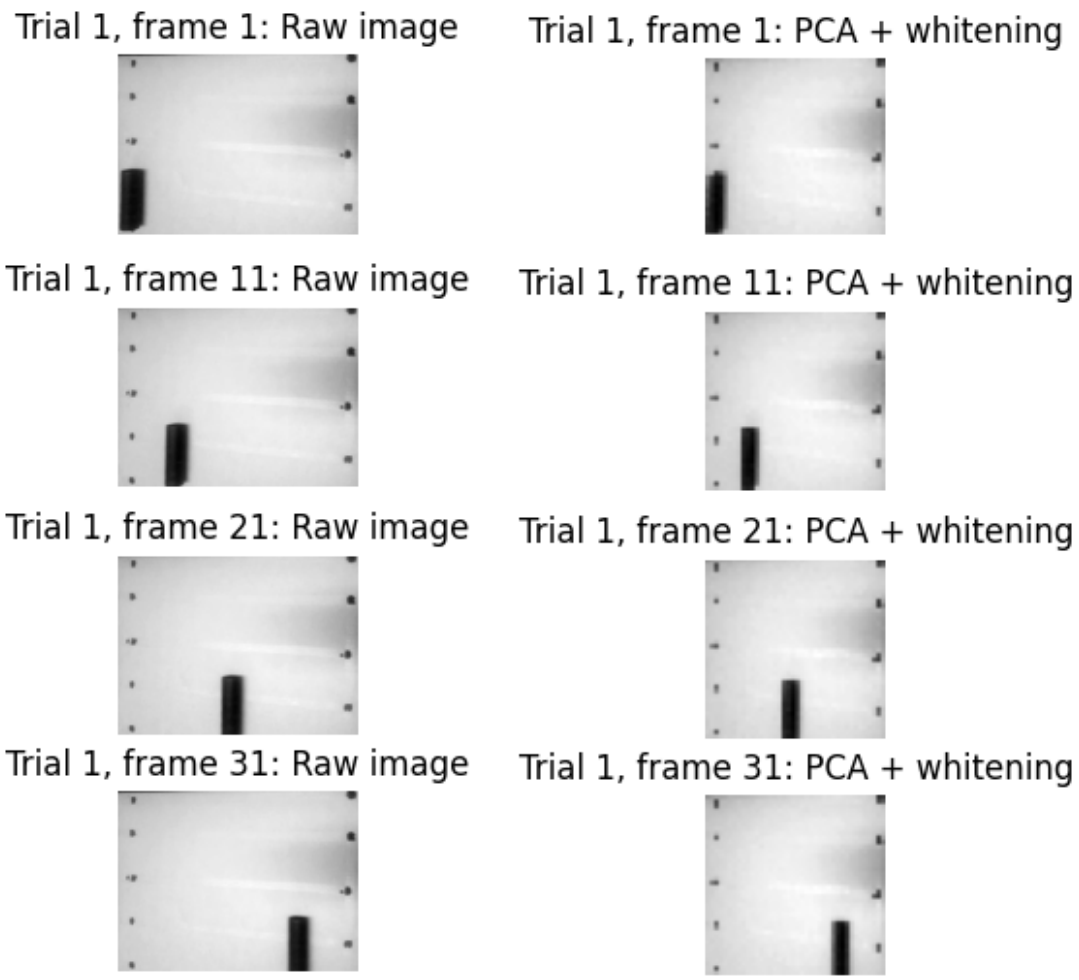

Figure 3.6: Sequence of 4 raw images, where a black cylinder object is moving rightwards (1^{st} column); image after pre-processing steps namely, conversion from RGB to greyscale, resizing, PCA and whitening (2^{nd} column).

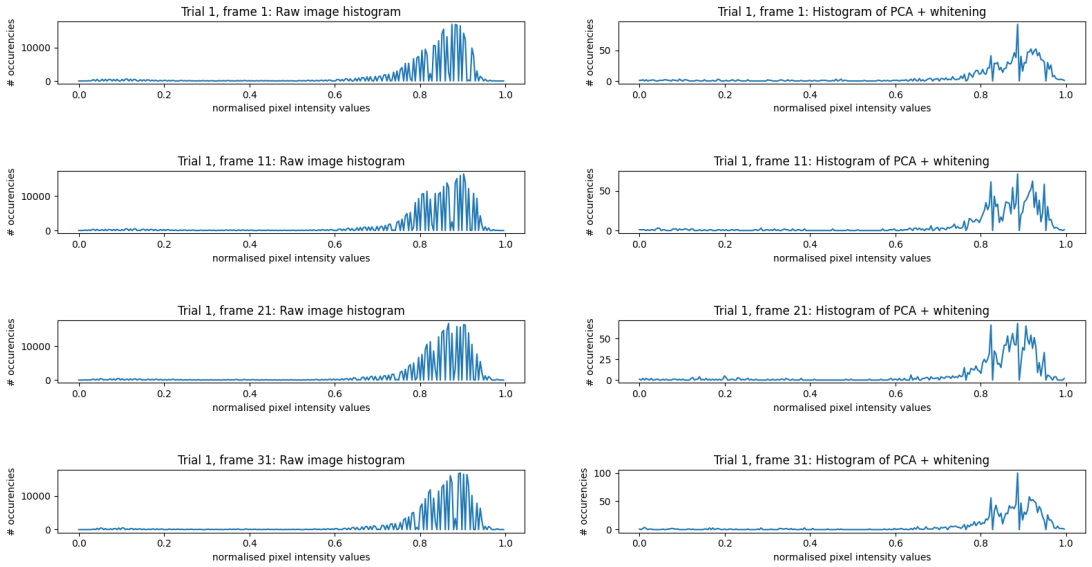

Figure 3.7: Histograms of the sequence of the images, shown in Figure 3.6. The histograms of pre-processed images are shown in the 1^{st} column and histograms of the post-processed images are shown in the 2^{nd} column.

Figure 3.7 depicts the histograms of the pre- and post-processed images. Compared with histograms from the first-column, the histograms from the second column show a dimensionality reduction due to using the PCA and, therefore, reducing the local correlation between pixels and speed performance.

The results labelled with MHSNNv1 were obtained when tested against the dataset after the pre-processing stage 1 (i.e., image conversion from colour to greyscale), whereas the MHSNNv2 results were obtained against the dataset after the pre-processing steps 1 and 2 (i.e., image conversion from colour to greyscale followed by the PCA whitening).

3.3.3 Simulation Process

The simulation was performed using the Brian2 SNN simulator[1].

Step 1 - Preparing the simulation: The image sequences were randomly split into four dataset subsets, each comprising different training and testing batches. A total of 75 batches of image sequences (2303 images) were used for training and the remaining 25 batches of image sequences (816 images) for testing per movement type. The batches of image sequences were all loaded into memory before starting the training/simulation.

Step 2 - Setting the simulation parameters: The simulation, neuron, and synapses parameters were set in accordance with the Brian 2 documentation [298]. The simulation was configured with a time step of $\tau = 1ms$. The following parameters were set for all neurons: $u_{reset} = -1mV$, $\tau_{refractory} = 0$. The threshold for neurons in Layer 1, 2 and 3 was set to $u_{th} = 0.0mV$ and for Layer 4 $u_{th} = 30.0mV$. The weights for Layers 1, 2 and 3 were set constant. The neurons in Layer 4 had their weights trained using the ReSuMe algorithm, with the following parameters: $\tau^d = 20ms$, $\tau^l = 20ms$, $A^d = 0.01$, $A^l = 0.01$ (for excitatory synapses) and $A^d = -0.01$, $A^l = -0.01$ (for inhibitory synapses).

[1]The Brian2 SNN simulator. http://brian2.readthedocs.io/en/stable/index.html, last accessed: 31/01/2018

Step 3 - Simulation stages: The simulation can be divided into two stages, namely training and testing.

- **Training Stage:** During the simulation training stage, a teacher signal is used to train each cell's weights. The weights of all synapses connected to all the Layer 4 neurons were initialised with 1.0. The training is repeated 10,000 times[1] on the training batches, while the weights are updated constantly.
- **Testing Stage:** The testing mode is repeated once on the test batches, while the weights are kept constant. In the test mode, ReSuMe is not used, and the outputs of Layer 4 are scored against the expected results.

3.3.4 Custom Object Direction Detection algorithms

The OpenCV's BS algorithms (i.e. MOG, MOG2, CNT, KNN, GMG, LSBP and GSOC) are highly efficient algorithms designed for modelling the dynamic background changes (i.e. about two hundred frames are required to train the background model) and classifying all the outliers as foreground. The CODD algorithm makes use of the OpenCV's BS algorithms for extracting the foreground from the background. The direction of the movement is obtained by computing the centre of mass (cM) from the foreground blob with

[1]The value 10,000 was obtained experimentally.

the coordinates (i, j) and comparing cM to the centre of mass from the previous frame (cM_{prev}) with the coordinates (i_{prev},j_{prev}). When detecting vertical motion, the object moves upwards when $cM(i) < cM_{prev}(i_{prev})$, remains stationary when $cM(i) = cM_{prev}(i_{prev})$ or moves downwards when $cM(i) > cM_{prev}(i_{prev})$. Similarly, when detecting horizontal motion, the object moves leftwards when $cM(j) < cM_{prev}(j_{prev})$, remains stationary when $cM(j) = cM_{prev}(j_{prev})$ or moves rightwards when $cM(j) > cM_{prev}(j_{prev})$. Algorithm 1 describes the CODD algorithm (see Figure 3.8).

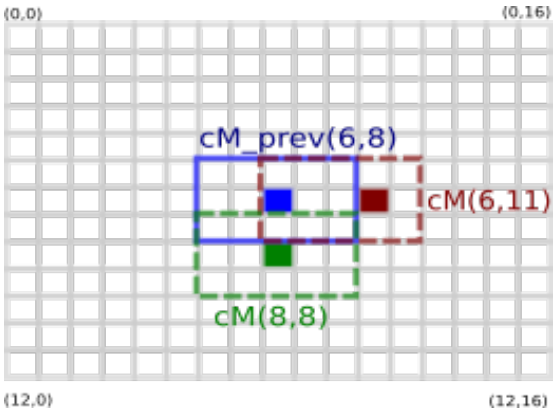

Figure 3.8: Detection of the object movement direction. The figure represents an image of 13×17 where a given object (blue rectangle) performs rightwards (dashed brown rectangle) and downwards (dashed green rectangle) movements. In the case of the brown rectangle, $cM(j) = 11$, $cM_{prev}(j_{prev}) = 8$ and therefore the object is moving rightwards because $cM(j) > cM_{prev}(j_{prev})$. While for green rectangle, $cM(i) = 8$, $cM_{prev}(i_{prev}) = 6$ and therefore, the object is moving downwards because $cM(i) > cM_{prev}(i_{prev})$.

Algorithm 1 CODD algorithm pseudocode

```
switch (backgound_subtraction_alg)
case MOG:
    bgsub = createOpenCVBackgroundSubtractorMOG
case MOG2:
    bgsub = createOpenCVBackgroundSubtractorMOG2
case CNT:
    bgsub = createOpenCVBackgroundSubtractorCNT
case KNN:
    bgsub = createOpenCVBackgroundSubtractorKNN
case GMG:
    bgsub = createOpenCVBackgroundSubtractorGMG
case LSBP:
    bgsub = createOpenCVBackgroundSubtractorLSBP
case GSOC:
    bgsub = createOpenCVBackgroundSubtractorGSOC
end switch
set_movement_type
for k:=calibration to classification do
    for i:=0 to number_images do
        get_image
        img = apply_bgsub
        if classification then
            binarise_img_and_normalise_img
            compute_cM
            if i = 0 then
                cM_prev = cM
            end if
            if movement_type = horizontal then
                direction = cM(j) − cM_prev(j_prev)
            else
                direction = cM(i) − cM_prev(i_prev)
            end if
            if direction = expected_direction then
                TP = TP + 1
            else
                FP = FP + 1
            end if
        end if
    end for
    if classification then
        store_results
    end if
end for
```

3.3.5 Metrics

The following metrics ruled by equations 3.13 and 3.13 were used to compute the PCC and Percentage of Wrong Classifications (PWC) using the True Positives (TP) and False Positives (FP).

$$PCC = \frac{TP}{TP+FP}.100 \tag{3.12}$$

$$PWC = \frac{FP}{TP+FP}.100 \tag{3.13}$$

3.4 Results

The voltage threshold of the Layer 1 neurons was obtained by computing the average of all the images used for training. The Brian2 simulator has a structure called "TimedArray" used to expose the images to the first layer of neurons. Images are converted into 1D vectors, and each image is a row of "TimedArray".

A vector of neuron indexes was used to retain the spatial information between image pixels and neurons and the desired connectivity between neurons. The Brian2 exposes each row (image) after the predefined time step (The MHSNN uses a 1ms time step).

Spike monitors were configured and used for tracking all the spikes generated by layer and plotted at the end of the simulation.

The synaptic weights were updated with the output generated by the ReSuMe training algorithm during each training mode iteration in the training mode.

3.4.1 Horizontal movement test

The results obtained from the horizontal test sequences are shown in Figures 3.9, 3.10 and 3.11.

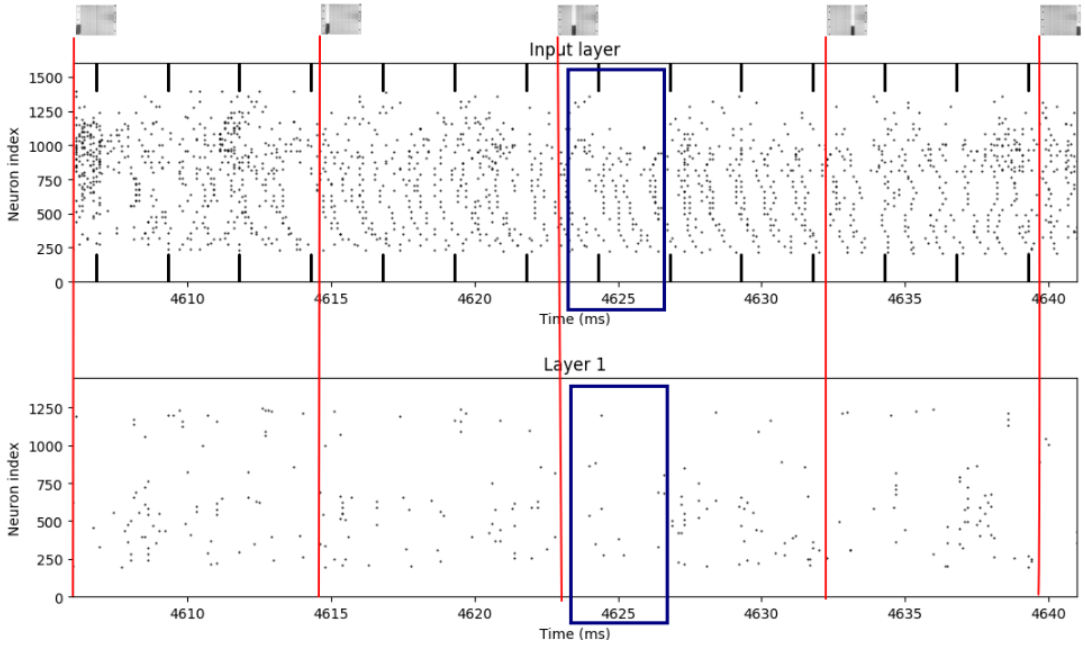

Figure 3.9: Raster plot of the spiking pattern obtained during the period [4605,4640]ms (a black cylinder object was moving rightwards) and generated by the input layer (after converting the graded values into spikes) and Layer 1 (edge extraction) neurons. The blue rectangle is used to track the spike events generated during the period [4624,4625]ms.

Referring to Figure 3.9, during the period [4605,4640]ms the image was performing a movement rightwards. The input layer illus-

trates the graded values after the conversion to spike events (values above 0.85 are considered spike events). Layer 1 shows the spike pattern generated from the edge extraction. The spiking pattern in Figure 3.9 was generated during test mode (after training).

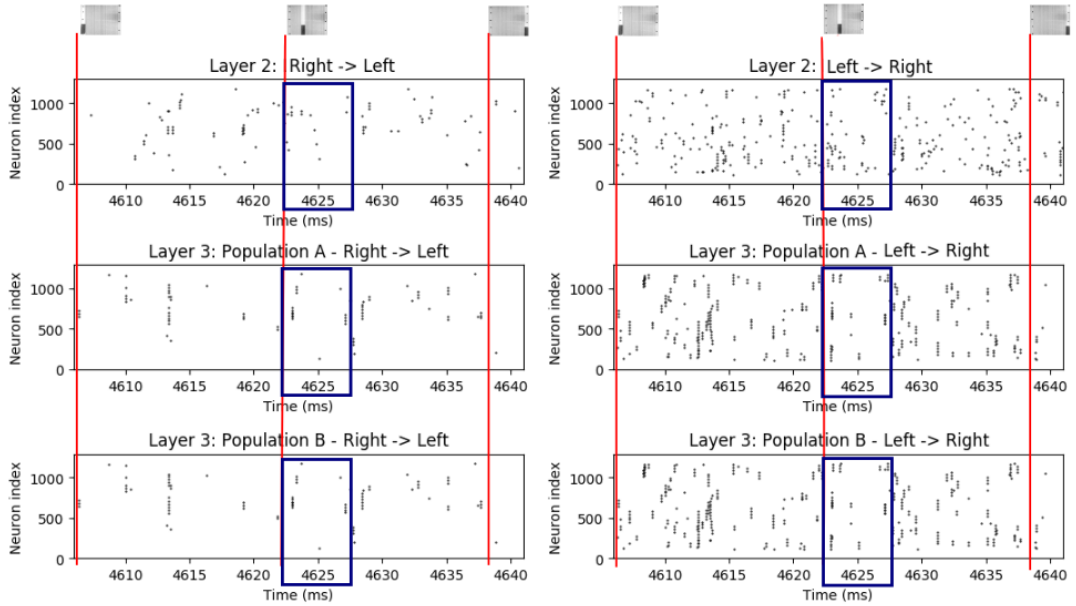

Figure 3.10: Raster plot of the spikes obtained during the period [4605,4640]ms (a black cylinder object was moving rightwards) and generated by the neurons in Layers 2 and 3. The blue rectangle is used to track the spike events generated during the period [4624,4625]ms.

Referring to Figure 3.10, the spike activity from the right cells is more prominent than the left cells, which is a consequence of the type of movement. The spike patterns obtained in population A ($frame[t] - frame[t-1]$) are very similar to the ones obtained from population B ($frame[t] - frame[t-2]$), which is a consequence of having slow movements that are detected by both populations of

neurons.

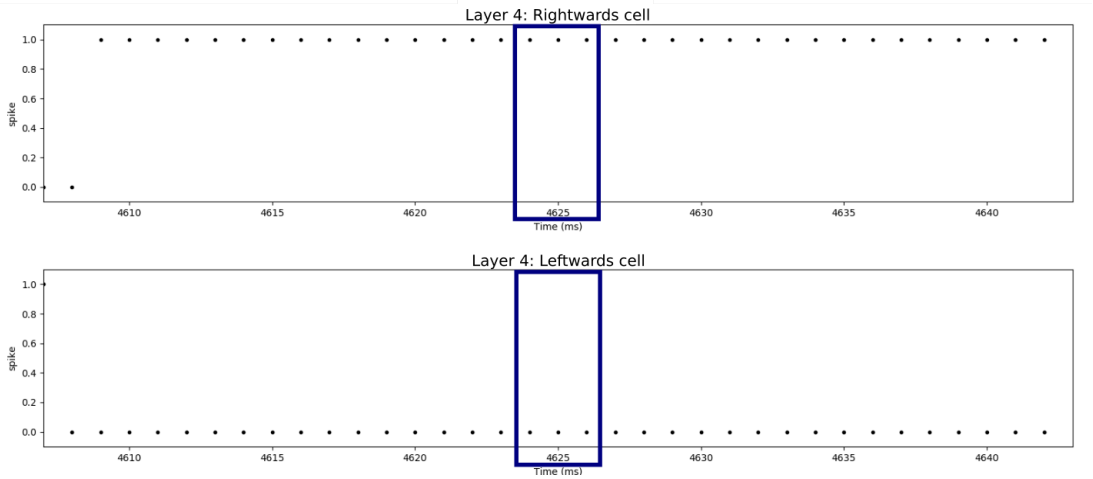

Figure 3.11: Raster plot of the spikes pattern obtained during the period [4605,4640]ms (a black cylinder object was moving rightwards) and generated by the horizontal sensitive cells. The blue rectangle is used to track the spike events generated during the period [4624,4625]ms.

Figure 3.11 shows that each cell is spiking at the expected time. The PWC was 7% for the horizontal cells. The scoring algorithm compares the output result with the expected result, and the error counter is increased by one every time an error is detected (i.e. false positives or false negatives). The error is associated with a sudden change of image sequences. This phenomenon occurs when the last image of a given batch is followed by the first image of a new batch. This situation triggers a different spike pattern compared with the previous spike patterns (Layer 3 populations A and B).

3.4.2 Vertical movement test

The results obtained from the vertical test sequences are shown in Figures 3.12, 3.13 and 3.14.

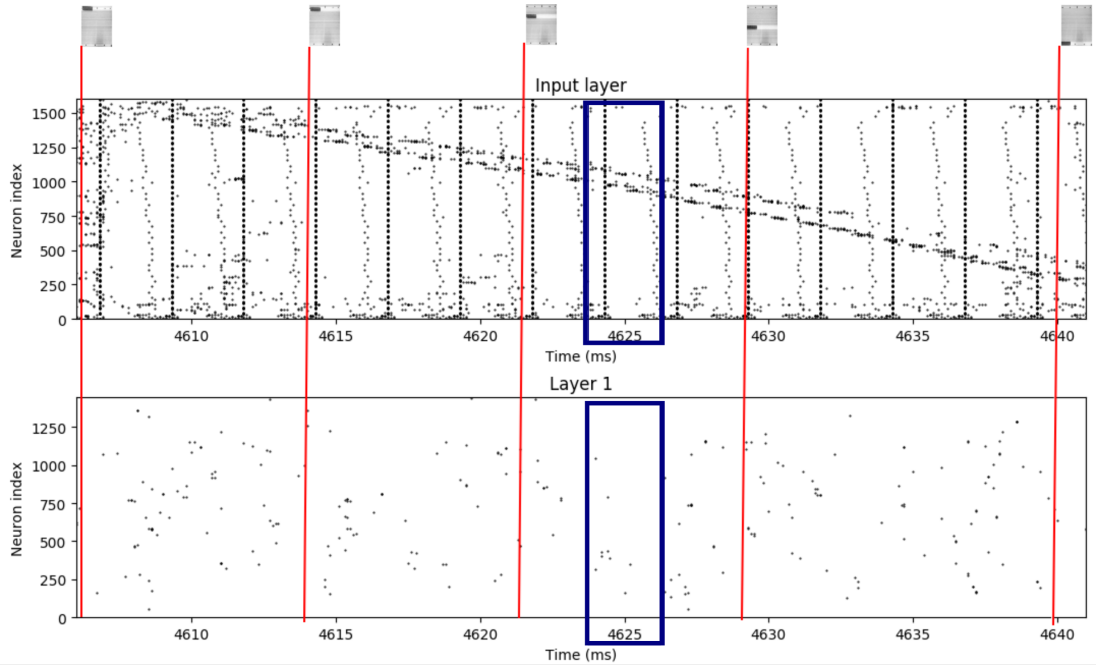

Figure 3.12: Raster plot of the spikes obtained during the period [4605,4640]ms of the vertical test and generated by the input layer (after converting the graded values into spikes) and Layer 1 neurons. The blue rectangle is used to track the spike events generated during the period [4624,4625]ms.

Referring to Fig. 3.12, during the period [4605,4640]ms the object is moving downward. The input layer shows the graded values after the conversion to spike events (values above 0.85 are considered spike events). In Layer 1 the spike pattern generated from the edge extraction is shown. The spiking pattern in Figure 3.12 is clearly distinct from the spike pattern in Figure 3.9. It is possible to infer

from the spike pattern of the input layer that the object is moving downward.

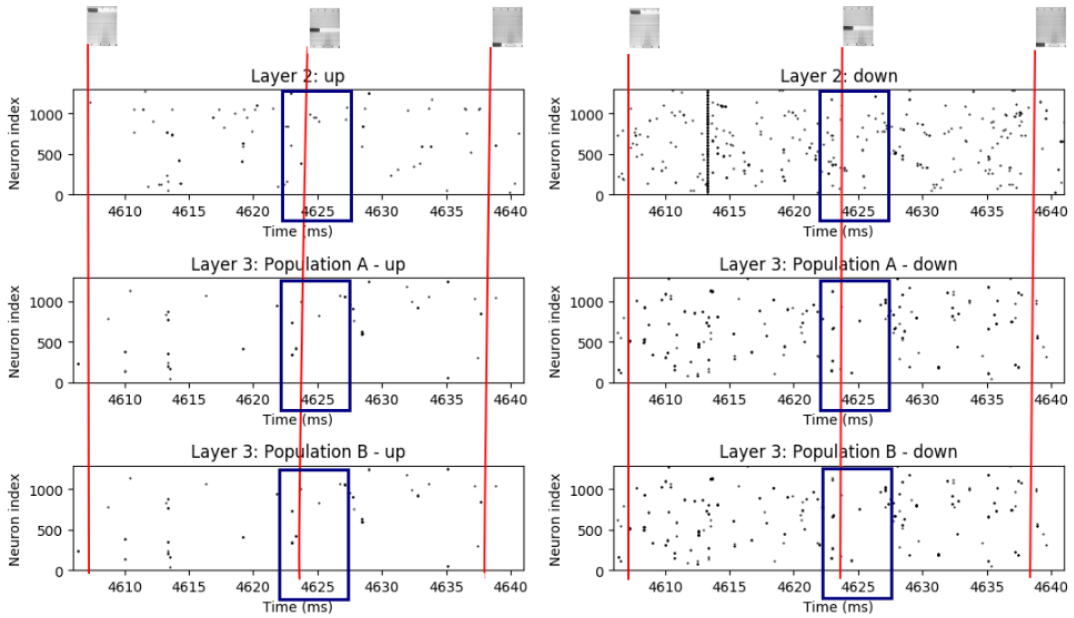

Figure 3.13: Raster plot of the spiking pattern obtained during the period [4605,4640]ms of the vertical test and generated by the neurons in Layers 2 and 3. The blue rectangle is used to track the spike events generated during the period [4624,4625]ms.

Referring to Fig. 3.13, the spike activity of the down cells is more prominent than the up cells, which is a consequence of the type of movement. Again, the spike patterns obtained from population A ($frame[t] - frame[t-1]$) are very similar to those obtained from population B ($frame[t] - frame[t-2]$), which is a consequence of having slow movements that are detected by both populations of neurons.

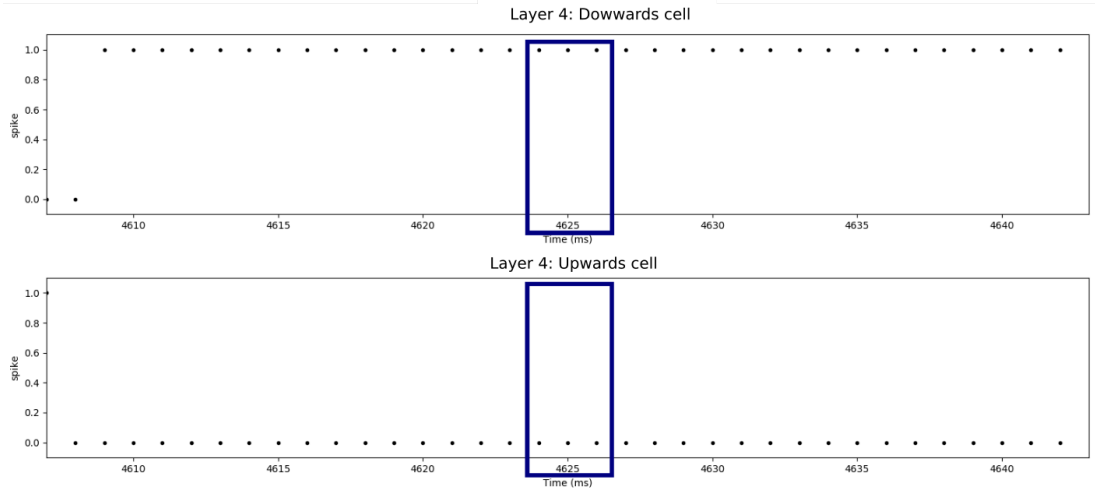

Figure 3.14: Raster plot of the spiking pattern obtained during the period [4605,4640]ms of the vertical test and generated by the vertical sensitive cells. The blue rectangle is used to track the spike events generated during the period [4624,4625]ms.

Figure 3.14 shows that each cell is spiking at the correct time because the first set of images are upwards movements and then downwards movements. It is also seen in Figure 3.14 that the right cell generated spike events, while the left cell did not generate spike events. The PWC was 6.5% for the vertical cells. The errors occur when the last image of a sequence is followed by the first image of a new sequence (i.e. when a new sequence starts and the object moves from the end position to the start position of two sequential images).

3.4.3 Results per category

The MHSNN was tested against the dataset before applying PCA whitening (MHSNNv1) and after applying PCA whitening (MHSNNv2) (see Section 3.3.1 for further details). Although the CODD variants (i.e. MOG, MOG2, CNT, GMG, KNN, LSBP, GSOC) were also tested against the dataset before applying PCA whitening, the PCC, PWC and processing results were the same for the dataset after applying PCA whitening. Table 3.1 shows leftwards movements classification and processing time per method.

Table 3.1 Leftwards movements classification and processing time per method.

Method	movement type	PCC	PWC	processing time
MHSNNv2	leftwards	93.6%	6.4%	1731.352 ms
CODD-MOG	leftwards	90.5%	9.5%	1.304 ms
CODD-KNN	leftwards	90.1%	9.9%	1.385 ms
MHSNNv1	leftwards	87.0%	13.0%	1760.069 ms
CODD-MOG2	leftwards	86.2%	13.8%	1.195 ms
CODD-CNT	leftwards	61.1%	38.9%	1.4 ms
CODD-GSOC	leftwards	46.1%	53.9%	3.427 ms
CODD-LSBP	leftwards	44.4%	55.6%	3.665 ms
CODD-GMG	leftwards	0.0%	100.0%	1.655 ms

The results show that the MHSNNv2 shows the best results in terms of highest PCC and lowest PWC when compared with the other CODD variants. Nevertheless, the CODD-MOG shows the second-best result with an average processing time of 1.304ms which is much faster than the MHSNNv2. It is also possible to infer that

the PCA whitening has contributed to an improvement of the MHSNN PCC by 6.6%.

Table 3.2 shows rightwards movements classification and processing time per method.

Table 3.2 Rightwards movements classification and processing time per method.

Method	movement type	PCC	PWC	processing time
MHSNNv2	rightwards	92.4%	7.6%	1775.007 ms
CODD-MOG	rightwards	93.1%	6.9%	1.262 ms
CODD-KNN	rightwards	92.5%	7.5%	1.716 ms
CODD-MOG2	rightwards	88.8%	11.2%	1.545 ms
MHSNNv1	rightwards	88.2%	11.8%	1803.407 ms
CODD-CNT	rightwards	61.1%	38.9%	1.451 ms
CODD-GSOC	rightwards	47.9%	52.1%	2.605 ms
CODD-LSBP	rightwards	45.1%	54.9%	3.38 ms
CODD-GMG	rightwards	0.0%	100.0%	1.604 ms

When compared to the other CODD variants, the results demonstrate that the MHSNNv2 exhibits the highest PCC and lowest PWC. Again, the MHSNNv2 is outperformed by the CODD-MOG, which has a processing time average of 1.262 milliseconds. It is also feasible to conclude that the PCA whitening has helped to improve the MHSNN PCC by 4.2 percent.

Table 3.3 shows downwards movements classification and processing time per method.

Table 3.3 Downwards movements classification and processing time per method.

Method	movement type	PCC	PWC	processing time
MHSNNv2	downwards	93.9%	6.1%	1898.088 ms
CODD-MOG	downwards	93.1%	6.9%	1.213 ms
CODD-KNN	downwards	92.3%	7.7%	1.491 ms
CODD-MOG2	downwards	88.8%	11.2%	1.228 ms
MHSNNv1	downwards	86.1%	13.9%	1928.457 ms
CODD-CNT	downwards	61.1%	38.9%	1.412 ms
CODD-GSOC	downwards	47.7%	52.3%	2.761 ms
CODD-LSBP	downwards	45.3%	54.7%	3.488 ms
CODD-GMG	downwards	0.0%	100.0%	1.642 ms

The results show that the MHSNNv2 exhibits the greatest PCC and lowest PWC values when compared to the other CODD variants. Once more, the CODD-MOG outperforms the MHSNNv2 with an average processing time of 1.213 milliseconds. It is also possible to extrapolate that the PCA whitening contributed to an improvement of the MHSNN's PCC by 7.8%.

Table 3.4 shows upwards movements classification and processing time per method.

Table 3.4 Upwards movements classification and processing time per method.

Method	movement type	PCC	PWC	processing time
MHSNNv2	upwards	93.1%	6.9%	1749.643 ms
CODD-MOG	upwards	90.5%	9.5%	1.2 ms
CODD-KNN	upwards	90.4%	9.6%	1.377 ms
CODD-MOG2	upwards	86.2%	13.8%	1.154 ms
MHSNNv1	upwards	87.6%	12.4%	1905.820 ms
CODD-CNT	upwards	61.1%	38.9%	1.446 ms
CODD-GSOC	upwards	46.1%	53.9%	2.749 ms
CODD-LSBP	upwards	44.3%	55.7%	3.557 ms
CODD-GMG	upwards	0.0%	100.0%	1.553 ms

The results indicate that, when compared to the other CODD variations, the MHSNNv2 exhibits the highest PCC and lowest PWC values. The CODD-MOG performs better than the MHSNNv2 once more, with an average processing time of 1.2 milliseconds. It is also reasonable to deduce that the improvement of the MHSNN's PCC by 5.5 percent was caused by the PCA whitening.

3.5 Discussion

The MHSNN architecture was designed to detect horizontal and vertical movements. The MHSNN detected leftwards, rightwards, downwards, and upwards movements in 93.6%, 92.4%, 93.9% and 93.1%, respectively when tested against the custom semisynthetic dataset. The PWCs were consequence of FPs at the beginning and

end of the image sequences because the sequences were collated together on a single *TimedArray*. The issue occurred because the first frame of another sequence was followed by the last frame of a given sequence. The MHSNN was one of the first SNN capable of detecting horizontal and vertical movements when tested on semisynthetic datasets at the time of its publication back in 2018 [293].

A CODD algorithm that combined the seven BS available in the OpenCV library was implemented to benchmark against the MHSNN. Although the MHSNN was ranked first in terms of the PCC and exhibited the lowest PWC detecting the motion direction when tested against the semisynthetic dataset, follow-up tests on natural datasets demonstrated that the MHSNN would have to be scaled up, which would increase the latency even more. Furthermore, the Brian 2 simulator has proven not to be suitable for real-time applications because i) all the dataset must be loaded into the memory before starting the simulation and ii) Brian 2 is not optimised for running large SNNs (i.e. above 17000 neurons and 173700 synapses). The speed limitations are very obvious when looking at the processing times of the MHSNNv2 in Tables 3.1, 3.2, 3.3, and 3.4.

The lessons learnt from the MHSNN architecture implementation

for targeting real-time applications using natural datasets were: 1) a better SNN simulator or a customised SNN simulator would be required to decrease substantially the latency, 2) robust object motion detection requires the combination of simpler theories BS algorithms with SNNs and 3) the custom SNN would most likely have to accommodate more than 100,000 neurons to provide the required accuracy. Finally, the MHSNN architecture was fundamental for designing the HSMD architecture, which will be discussed in the next chapter 4.

Chapter 4

HSMD: Hybrid Spiking Motion Detection

The HSMD algorithm proposed in this chapter was designed to detect object motion in real-time and therefore overcome multi-hierarchical spiking neural network speed limitations. The speed limitation was dealt by replacing the Brian 2 simulator with an efficient and parallel C++ implementation and through reducing both the number of layers and synapses. Furthermore, the HSMD combines a BS algorithm with a customised SNN for detecting object motion as opposed to detecting the direction of motion like in the MHSNN. The HSMD enhances the GSOC BS algorithm with a customised 3-layer SNN that outputs spiking responses akin to the OMS-GC. The algorithm was compared against existing BS approaches available on the OpenCV library, specifically on the CD-

net2012 and the CDnet2014 benchmark datasets. The results show that the HSMD was ranked overall first among the competing approaches and has performed better than all the other algorithms in four of the categories across all the eight test metrics. Furthermore, the HSMD proposed in this chapter is the first to use an SNN to enhance an existing state-of-the-art GSOC BS algorithm, and the results demonstrate that the SNN provides near real-time performance in realistic applications.

4.1 Introduction

In computer vision, object motion detection is traditionally performed using BS methods, where the foreground (pixels or group of pixels whose light intensity values have suffered an abrupt variation) are compared with the previous image or background model [16; 17; 126; 299]. BS are algorithms for extracting the background from the foreground by modelling the background through the comparison of the current frame with previous frames [300; 301; 302; 303; 304; 305]. BS methods can be implemented using 1) mathematical-based, 2) machine learning, 3) signal processing, and 4) DNNs approaches [16; 17]. Mathematical-based approaches are the simplest way to model backgrounds using temporal average, temporal median, and histograms, which can be improved using

refined models (such as a mixture of Gaussians, kernel density estimation, etc.) and require low computational resources [16]. Machine learning models are more robust for performing BS, but they must be trained on the target visual features and require significant computational resources [17]. Signal processing models are used to model the background using the temporal history of pixels as 1D signals and usually require moderate computational resources [16]. DNN models are by far the most accurate, but they are also the most computationally intensive and therefore not suitable for real-time applications. Although less robust, the classical mathematical BS models are better suited for real-time applications. As real-time processing is a key objective of this work, we focus only on mathematical models in this chapter.

BS methods can be classified as 1) Mathematical, 2) Machine Learning and 3) Signal processing [16; 17]. Mathematical theories are the simplest way to model backgrounds using temporal average, temporal median and histograms, which can be improved using refined models (such as a mixture of Gaussians, kernel density estimation, etc.) and require low computational resources [16]. Machine learning models are more robust for performing BS, but they must be trained on the target visual features and require significant computational resources [17]. Signal processing models used to model the

background using the temporal history of pixels as 1D signals and usually require moderate computational resources [16]. Although less robust, the classical mathematical BS models are better suited for real-time to near real-time applications. As real-time processing is a key objective of this work, we focus only on mathematical models in this chapter.

The OpenCV library [306] is one of the most robust and reliable computer vision libraries, maintained by a wide Open Source community (including high profile companies such as Intel, Microsoft and Google) and is the reference library for computer or robot vision researchers [31]. The OpenCV's BS algorithms (i.e. MOG, MOG2, CNT, KNN, GMG, LSBP and GSOC) are highly efficient BS algorithms that were designed for modelling the dynamic background changes (i.e. about two hundred frames are required to train the background model) and classifying all the background outliers as foreground. The GSOC algorithm was selected to perform the first stage of BS over the other BS available on the OpenCV library because it has demonstrated better accuracy on the CDnet2012 and CDnet2014 datasets [30] when compared to other algorithms available on the OpenCV library.

The HSMD model reported in this chapter was inspired by the object motion functionality exhibited by vertebrate retinas, in which OMS-GC determine the difference between a local patch's motion trajectory and the background [18]. In fact, the HSMD is an improved version of GSOC BS algorithm [30; 148] is enhanced by a 3-layer SNN, forming a hybrid architecture.

The main contributions of the work reported in this chapter are i) an object motion detection model inspired by the OMS-GC designed to work with COTS cameras, ii) enhancement of the dynamic BS (mathematical model) using the 3-layered SNN and iii) optimisation of the proposed method for processing live capture feeds in near real-time. The algorithm was tested on the CDnet2012 [9] and CDnet2014 benchmark datasets [32] and compared with the OpenCV's BS algorithms (i.e. MOG, MOG2, CNT, KNN, GMG, LSBP and GSOC). The HSMD can detect motion using commercial-off-the-shelf camera feeds and/or video clips using SNN, as opposed to cameras exploiting dedicated custom architectures.

This chapter is structured as follows: the HSMD is described in section 4.2; the training details, use-case scenarios and HSMD parameterisation are described in section 4.3; the results are reported

and analysed in section 4.4; and the discussion and future work are discussed in section 4.5.

4.2 HSMD architecture

The HSMD is a combined BS/SNN network to create a hybrid model for detecting motion, emulating the elementary functionalities of the OMS-GC as described in [18].

The architecture of the HSMD is shown in Figure 3.2. There are five layers to the overall architecture. Layer 1 performs the BS using the GSOC algorithm. The resulting BS frames are fed into Layer 2 of the SNN, where the pixel intensity values are converted into currents that are proportional to the light intensity (see 4.2.2). The BS-converted currents are fed to the Layer 2 neurons via a 1:1 synaptic connectivity. Layer 2 neurons are synaptically connected to Layer 3 neurons, which perform the first stage of motion analysis; Layer 3 neurons connect to Layer 4 neurons via 1:1 synaptic connectivity. Layer 4 neurons perform precise motion detection. A median filter is used to filter random spikes related to local random illumination variations.

The LIF was the spiking neuron model used in this work because of its simplicity, computational efficiency and suitability for

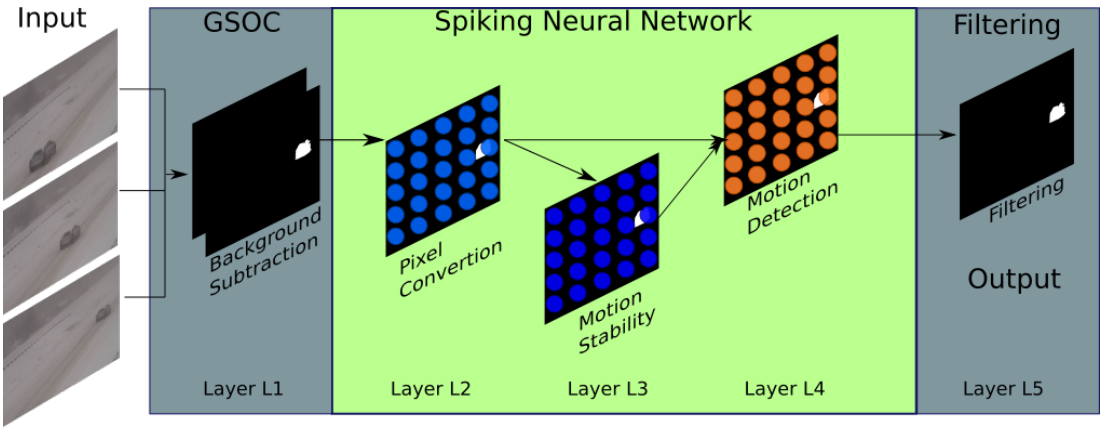

Figure 4.1: HSMD with (i) $n \times m$ image input followed by the BS using the GSOC algorithm, three spiking neuronal layers and filtering. Layer 1: BS, Layer 2: pixel intensity to spike events encoding, Layer 3: Motion stability, Layer 4: motion detection and Layer 5: filtering.

processing images in near real-time. The LIF spiking neuron model exhibits similar, but less complex, dynamics compared to real biological neurons (see Figure 2.4) [4]. The LIF neuron's dynamics are described by equation 2.1.

4.2.1 Input Layer: background subtraction and reduction

Each $n \times m$ image frame (i.e. camera, video sequence or image sequences) is converted into greyscale.

The GSOC [147] delivers a dynamic and adaptive BS using colour descriptors and various stabilisation heuristics [30; 148] while processing the frames pixel-wise and leveraging the parallelism inside OpenCV [30].

4.2.2 Layer 2: Pixel intensities values to currents encoding

Pixel intensity values are converted into proportional currents and fed into the spiking neurons in Layer 2 via a 1:1 connectivity. The Layer 1 neurons were trained to trigger spike events proportional to the pixel intensity values, as described by equation 4.1.

$$i_c(x, y) = I(x, y).c \tag{4.1}$$

where $i_c(x, y)$ is the corresponding current for the image light intensity value $I(x, y)$ at coordinates x and y, and c is a conversion constant obtained experimentally (in our case, c=17.5).

4.2.3 Layer 3: Motion stability

Layer 3 is used to perform motion stabilisation through the creation of local buffers by delaying the propagation of spike events. A delay is created when a given neuron of layer 2 connects to a neuron in layer 3, before being passed to Layer 4, instead of intra-layer connectivity between direct Layer 2 and Layer 4. Spike events passing through Layer 3 are buffered by neurons in Layer 3 for one simulation time-step (δt, in this work, δt = 10 ms) and presented to the neurons in Layer 4. N[n] in the following simulation time-step.

The neurons in Layer 2 are connected to the Layer 3 neurons via a 1:1 connectivity. Finally, the Layer 3 neurons connect to the Layer 4 neurons via a 1:1 connectivity, as shown in Figure 4.2. All synaptic weights from Layer 2 to Layers 3 and 4 have a value of 1370 (obtained experimentally).

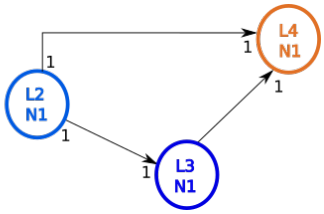

Figure 4.2: HSMD connectivity. In this example, it can be seen that the neuron 1 (N1) of each layer connects to the N1 of the subsequent layer.

4.2.4 Layer 4: Motion detection

The Layer 4 neurons receive synaptic connections from the neurons in Layer 2 and Layer 3 via excitatory synapses and exploit these spiking events to detect motion. Spike events generated by Layer 4 neurons resulted from dynamic changes between sequential image frames. Signals received directly from Layer 2 neurons enable detection of changes between the current image *frame n* and the previous image *frame n-1*. In contrast, those routed via Layer 3 neurons compare the image *frame n-1* with the image *frame n-2*. Layer 4 spike events are mapped into the corresponding area in the original image captured from the camera. The synaptic weights ob-

tained experimentally are 1370 for all the synapses. The Layer 2 to Layer 4 weights were tuned to forward all the spike events generated in Layer 2. The Layer 3 to Layer 4 synaptic weights were tuned to produce spike events from the Layer 4 neurons for each group of two sequential spike events. The main goal is to give high importance (larger weight) to new spike events ($frame[n] - frame[n-1]$) and lower importance to older spike events ($frame[n-1] - frame[n-2]$).

4.2.5 Layer 5: Filtering

The Layer 4 neurons' spike events matrix is mapped into a motion matrix M_d of the same size as the captured image (i.e. $n \times m$). The events in the M_d matrix are filtered using an averaging filter described by equations 4.2 and 4.3:

$$H(u,v) = \frac{1}{u.v}\left(\begin{bmatrix} w_{0,0} & \dots & w_{0,u} \\ \dots & \dots & \dots \\ w_{v,0} & \dots & w_{v,u} \end{bmatrix}\right) \tag{4.2}$$

$$Y_d(x,y) = M_d(x,y) * H(u,v) \tag{4.3}$$

where $Y_d(x,y)$ is the filtered motion detection matrix, $H(u,v)$ is the averaging Gaussian filter [294], u and v are the convolution window length and height respectively, $*$ is the convolution operator, w is

the filter window.

4.3 Implementation details

The HSMD was implemented in C++ using the C++ Standard Template Library (STL) 17 (implementation of data structures) [307], Boost 1.71 (file management) [308] and OpenCV 4.5.0 (image processing) [306]. The decision not to use any existing SNN simulator (such as Brian 2 [298], and NEST [53]) was made to ensure that the SNN could have the lowest latency possible. The STL C++17 library offers a collection of C++ algorithms that have been optimised to deliver a higher degree of parallelisation when running on multicore CPU systems; The Boost library offers a wide range of reusable algorithms, including file management, time monitoring, and exception handling algorithms; and the OpenCV library offers a collection of algorithms for computer vision applications, including image manipulation and filtering, and BS algorithms.

4.3.1 HSMD setup

The HSMD initial setup includes the following steps:

Step 1 - Select between live capture, video analysis or image sequences: The user can opt to run the algorithm directly

on images being captured by the camera or provide the path of a video or set of image sequences for motion analysis.

Step 2 - Create the Layer 2 to Layer 4 neural network: Read the first image and compute the size of the image. The number of neurons is computed automatically from the dimensions of the first image in a sequence of images.

Step 3 - Set the neuronal parameters: The LIF parameters recommended in the references [309; 310] were used to configure the SNN. Therefore, the simulation was configured with a time step of δt=10 ms and the default neuron parameters as follows: initial V_m=-55.0 mV, E_L = -55.0 mV, C_m = 10.0 pF, R_m=1.0 MOhm, V_{reset}=-70.0 mV, V_{min}=-70.0 ms, V_{th}=-70.0 mV, τ_m=10.0 ms, t_{ref}=2 ms, w_{syn} = 1555.0 (neurons L3 and L4) and w_{p2i}=8.0 (L2 neurons only).

Step 4 - Start the image acquisition : Images are loaded from folders with sequences of images while the HSMD algorithm is being executed. The pseudo-code of the main algorithm is described in Algorithm 2.

Algorithm 2 HSMD main algorithm pseudo-code

```
1: newImage = get_new_image
2: newImageGrey = colour2grey(newImage)
3: set_number_neurons_from_newImageGrey_shape
4: build_neuronal_network
5: load_pretrained_weights
6: while frames available do
7:    reset_spike_events
8:    newImage = capture_image_camera
9:    newImageGrey = colour2grey(newImage)
10:   newImageReduced = newImageGrey
11:   dynSubImage = newImageReduced − previousImage
12:   previousImage = newImageReduced
13:   for I in dynSubImage do
14:      if dynSubImage[I]< Threshold then
15:         dynSubImage[I] = 0.0
16:      end if
17:      currents = convPixel2Current(dynSubImage)
18:      for i:=0 to timestep do
19:         apply_currents_to_neurons_L2
20:         compute_L2_neuron_spikes;
21:         convert_L2_neuron_spikes_to_currents;
22:         compute_L3_neuron_spikes;
23:         convert_L3_neuron_spikes_to_currents;
24:         compute_L4_neuron_spikes;
25:      end for
26:      spikes = get_sumSpikeEventsPerL4Neuron()
27:      masked_spikes = applyAveragingFilter(spikes)
28:      spikes = normalise(spikes)
29:      display(newImage)
30:      display(spikes)
31:   end for
32: end while
33: Display_spike_rates;
```

4.3.2 Datasets and metrics

4.3.2.1 Datasets

The CDnet2012 [9] (cited more than 379 times and CDnet2014 [32] (cited more than 300 times) benchmark datasets were designed for benchmarking BS algorithms. While the HSMD algorithm has been designed as an object detection algorithm and not a BS algorithm, nevertheless these two datasets provide challenging scenarios for robust comparable assessment of the proposed algorithm and network. The HSMD was compared against state-of-the-art BS algorithms available on the OpenCV library: MOG [131], MOG2 [141], KNN [142], GMG [145], LSBP [146], CNT [143] and GSOC [148]. The OpenCV BS algorithms were used because they are highly optimised, reliable, and publicly available to anyone who wants to test or compare their algorithms.

Each of the benchmark videos in the CDnet2012 [9] and CDnet2014 [32] fall into one or more of the challenge categories listed below:

CDnet2012 and CDnet2014

- **Baseline**: reference videos, which are relatively simple to classify; some videos contain very simple movements from the next four categories.

- **Dynamic Background**: videos that have both foreground and background motion (e.g. water movement and shaking trees).
- **Camera Jitter**: videos captured with cameras installed on unstable structures.
- **Shadow**: videos containing narrow shadows from solid structures or moving objects.
- **Intermittent Object Motion** videos include objects that are static for most of the time and suddenly start moving.
- **Thermal**: videos that exhibit thermal artefacts (i.e. bright spots and thermal reflections on windows and floors).

CDnet2014 only

- **Challenging Weather**: outdoor videos recorded during winter storm conditions with extremely low visibility.
- **Low Frame-Rate**: videos captured at frame rates ranging from 0.17 to 1 fps;
- **Night**: includes traffic videos with low visibility and strong headlights.
- **Pan, Tilt and Zoom (PTZ)**: videos recorded with cameras that were subjected to PTZ movements.

- **Air Turbulence**: videos filmed from distances of 5 to 15 km exhibiting air turbulence and frame distortion.

Table 4.1 lists the categories per each dataset.

Table 4.1 Categories available per each dataset

Category / dataset	bad weather	baseline	camera jitter	intermittent object motion	low frame rate	night videos	Pan, Tilt Zoom	Shadow	thermal	turbulence
CDnet2012		X	X	X				X	X	
CDnet2014	X	X	X	X	X	X	X	X	X	X

The BS algorithms were configured using the default OpenCV settings [306] and compared against the HSMD algorithm. The ground-truth provided by the datasets is composed of the following labels [9; 32]:

- **Static** - greyscale value 0;
- **Shadow** - greyscale value 50;
- **non-ROI** - greyscale value 85;
- **Unknown** - greyscale value 170;
- **Moving** - greyscale value 255;

The *static* and *moving* classes contain pixels that belong to the background and foreground, respectively; the *shadows*, one of the most challenging artefacts, should be classified as part of the background. The *unknown* region should not be considered either back-

ground or foreground because it contains pixels that cannot be accurately classified as background or foreground. The non-ROI pixels serve to exclude frames from being classified because some BS algorithms require several pixels for the model to stabilise (i.e. create the background model) and for preventing corruption by non-related activities to the considered category [9; 32]. Figure 4.3 shows the 5 class regions.

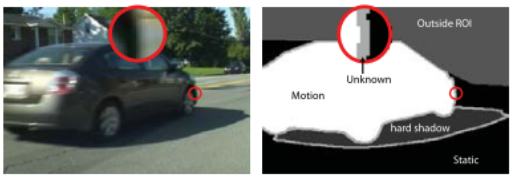

Figure 4.3: Raw image frame (left) and its respective ground-truth (right). The ground-truth images show the annotations using the datasets labels. Adapted from [9]

.

4.3.2.2 Metrics

The average performance obtained for each category using each BS method and the HSMD algorithms is characterised via eight metrics, as shown below. The four fundamental qualitative metrics are: TP, True Negatives (TN), FP and False Negatives (FN) [9; 32].

1. Recall (Re): $Re = \frac{TP}{TP+FN}$

   Re: ranked by **descending order**;

2. Specificity (Sp): $Sp = \frac{TN}{TN+FP}$;

Sp ranked by **descending order**;

3. False Positive Rate (FPR): $FPR = \frac{FP}{FP+TN}$;

   FPR ranked by **ascending order**;

4. False Negative Rate (FNR): $FNR = \frac{FN}{FN+TP}$;

   FNR ranked by **ascending order**;

5. Wrong Classifications Rate (WCR): $WCR = \frac{FN+FP}{TP+FN+FP+TN}$;

   WCR ranked by **ascending order**;

6. Correct Classifications Rate (CCR): $CCR = \frac{TP+TN}{TP+FN+FP+TN}$;

   CCR ranked by **descending order**;

7. Precision (Pr): $Pr = \frac{TP}{TP+FP}$;

   Pr ranked by **descending order**;

8. F-measure (F1): $F1 = 2 \times \frac{Pr.Re}{Pr+Re}$

   $F1$ ranked by **descending order**;

R: $R = \frac{\overline{Re}+\overline{Sp}+\overline{FPR}+\overline{FNR}+\overline{WCR}+\overline{CCR}+\overline{F1}}{nMet}$;

R ranked by **ascending order**;

RC:

$\overline{RC} = \frac{Re+Sp+FPR+FNR+WCR+CCR+F1}{nMet}$;

$\overline{RC}$ ranked by **ascending order**;

where $nMet$ is the number of metrics (8 in this case).

4.4 Results

The HSMD was tested on both datasets under the same conditions to ensure an accurate and rigorous comparison. The results are presented both as overall results and per category to better understand the specific performances obtained per method. The overall results for each method are presented in section 4.4.1 and the results per method and category are presented in section 4.4.2.

The ↑ indicates that the highest score is the best result, and the ↓ that the lowest result is the best result in the tables (4.2 to 4.5). The best results are highlighted using light grey for all the methods except the HSMD results, which are highlighted in dark grey. *Re* stands for recall, *Sp* specificity, FPR False Positive Rate, FNR False Negative Rate, WCR Wrong Classifications Rate, CCR Correct Classifications Rate, Pr Precision, F1 F-score, R Average Ranking and $\overline{RC}$ Average Ranking across all categories.

The results for each of the eleven categories shared by both CDnet2012 and CDnet2014 are shown in Figure 4.4.

4.4.1 Overall results

Tables 4.2 and 4.3 present the overall results obtained per method and per metric, ranked by $\overline{RC}$ (average ranking across all categories,

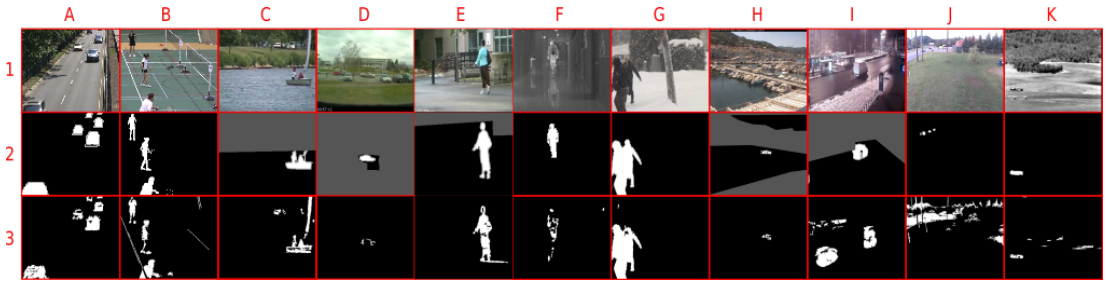

Figure 4.4: Results obtained for each of the eleven of the five categories (columns A to F) are common to both CDnet2012 and CDnet2014 datasets, while the remaining six categories (columns G to K) are only available on CDnet2014 dataset. Column A: baseline; B: camera jitter; C: dynamic background; D: dynamic object motion; E: shadow, F: thermal, G: bad weather, H: low frame rate; I: night videos, J: PTZ and K: turbulence. Row 1: RGB image; 2: ground-truth; and 3: HSMD binarised. The raw images, shown in the first row, demonstrate the scenarios that can be found in both datasets. The corresponding ground truth images, presented in the second row, show the 5 labels, namely, i) static [greyscale value 0], ii) shadow [greyscale value 50], iii) non-ROI [greyscale value 85], iv) unknown [greyscale value 170] and v) moving [greyscale value 255]. The corresponding binarised images generated by the HSMD are shown in the third row.

first column) in ascendant order.

The HSMD algorithm ranks in first place across all eight methods with which it is compared when tested against the CDnet2012 dataset (see the $\overline{RC}$ results in Table 4.2). Although the HSMD performed very well in five of the eight metrics, it is essential to highlight the results from the WCR, CCR, and F1 metrics. The results show that the HSMD is sensitive to object motion due to the highest correct counts and lowest wrong count rates that contributed to get the highest F-score and the second-best precision. Furthermore, the GSOC algorithm ranks in second place immediately after the HSMD.

Table 4.2 CDnet2012 overall results. Results ordered in descendent order by $\overline{RC}$

Method	$\overline{RC}$ ↓	Re ↑	Sp ↑	FPR ↓	FNR ↓	WCR ↓	CCR ↑	F1 ↑	Pr ↑
HSMD	**2.8**	0.52	0.994	0.006	0.23	**0.024**	**0.976**	0.77	0.62
GSOC	3.5	0.54	0.993	0.007	0.25	*0.024*	*0.976*	0.75	*0.63*
MOG2	3.8	0.37	0.995	0.004	0.24	0.026	0.974	0.76	0.50
GMG	3.9	0.20	*0.998*	*0.002*	*0.21*	0.033	0.967	*0.79*	0.32
KNN	4.3	0.39	0.995	0.005	0.26	0.025	0.975	0.74	0.51
MOG	4.5	0.32	0.996	0.004	0.26	0.030	0.970	0.74	0.44
CNT	6.1	*0.73*	0.927	0.073	0.71	0.081	0.919	0.29	0.41
LSBP	7.3	0.57	0.90	0.096	0.80	0.109	0.891	0.20	0.29

↑: the highest score is the best.
↓: the lowest result is the best.
Best HSMD results are highlighted using dark grey, while best results per category are highlighted with light grey for other methods. *Re* stands for Recall, *Sp* Specificity, FPR False Positive Rate, FNR False Negative Rate, WCR Wrong Classifications Rate, CCR Correct Classifications Rate, Pr Precision, F1 F-score and $\overline{RC}$ Average Ranking across all Categories.

Table 4.3 CDnet2014 overall results. Results ordered in descendent order by $\overline{RC}$

Method	$\overline{RC}$ ↓	Re ↑	Sp ↑	FPR ↓	FNR ↓	WCR ↓	CCR ↑	F1 ↑	Pr ↑
HSMD	**2.9**	0.55	0.993	0.007	0.35	0.018	0.982	0.65	**0.60**
GSOC	3.0	0.40	0.995	0.005	0.38	*0.017*	*0.983*	0.62	0.48
KNN	3.5	0.34	0.996	0.004	*0.32*	0.019	0.981	*0.68*	0.45
GMG	4.3	0.24	*0.997*	*0.003*	0.36	0.022	0.978	0.64	0.35
MOG	4.4	0.58	0.991	0.009	0.39	0.019	0.981	0.61	0.60
MOG2	4.5	0.39	0.994	0.006	0.42	0.018	0.982	0.58	0.47
LSBP	6.5	0.58	0.945	0.055	0.79	0.064	0.936	0.21	0.31
CNT	7.0	*0.72*	0.930	0.070	0.80	0.075	0.925	0.20	0.32

↑: the highest score is the best.
↓: the lowest result is the best.
Best HSMD results are highlighted using dark grey, while best results per category are highlighted with light grey for other methods. *Re* stands for Recall, *Sp* Specificity, FPR False Positive Rate, FNR False Negative Rate, WCR Wrong Classifications Rate, CCR Correct Classifications Rate, Pr Precision, F1 F-score and $\overline{RC}$ Average Ranking across all Categories.

Table 4.3 shows that the HSMD algorithm was ranked in first place in the average ranking $\overline{RC}$ across all 11 categories when tested

on the CDnet2014 dataset. The HSMD performed very well in seven of the eight metrics, and exceptionally well in the precision metric.

Table 4.4 HSMD overall results. Results ordered in descendent order by $\overline{RC}$

Dataset	$\overline{RC}$ ↓	Re ↑	Sp ↑	FPR ↓	FNR ↓	WCR ↓	CCR ↑	F1 ↑	Pr ↑
CDnet2012	**2.8**	0.52	0.994	0.006	0.23	0.024	0.976	0.77	0.62
CDnet2014	2.9	0.55	0.993	0.007	0.35	0.018	0.982	0.65	0.60

↑: the highest score is the best.
↓: the lowest result is the best.
Best results are highlighted in gray.
Re stands for Recall, *Sp* Specificity, FPR False Positive Rate, FNR False Negative Rate, WCR Wrong Classifications Rate, CCR Correct Classifications Rate, Pr Precision, F1 F-score and $\overline{RC}$ Average Ranking across all Categories.

Table 4.4 shows that there was a slight decrease in the HSMD performance when tested on the eleven categories available in the CDnet2014 as compared to the original six in the CDnet2012 dataset. Furthermore, the extra 6 categories, which are only present in the CDnet2014 contributed to an increase from 2.8 in the CDnet2012 to 2.9 in the CDnet2014 of $\overline{RC}$ across all categories. The increase of $\overline{RC}$ is justifiable with the degradation of 5 metrics, namely, Sp, FPR , FNR, F1 and Pr as a consequence of the complexity introduced by the video sequences from the extra six categories. This is more evident when analysing the individual Average Ranking (R) results per dataset as per listed in Table 4.5, and Figures 4.5 and 4.6.

Finally, it was anticipated that none of the methods would have excellent performance across all metrics because of the complexity

and size of the CDnet2012 and CDnet2014 datasets.

4.4.2 Results obtained per category

The R for each of the methods per category is shown in Table 4.5.

Table 4.5 Results per category. Results ordered in descendent order by R

IntObjMotion		shadow		cameraJitter		badWeather		dynamicBackground		nightVideos		PTZ		thermal		baseline		lowFramerate		turbulence	
Method	R↓	Method	R↓	Method	R↓	Method	R↓	Method	R↓	Method	R↓	Method	R↓	Method	R↓	Method	R↓	Method	R↓	Method	R↓
HSMD	**3.875**	*MOG2*	*2.375*	*LSBP*	*1.875*	*CNT*	*2.125*	*LSBP*	*1.75*	**HSMD**	**2.625**	*CNT*	*1.75*	**HSMD**	**3.25**	*MOG2*	*2.625*	*GSOC*	*2.25*	**HSMD**	**2.875**
LSBP	4.125	CNT	3.5	CNT	2.625	**HSMD**	**2.5**	MOG	2.375	LSBP	3.625	MOG	3.0	MOG	3.375	MOG	3.125	GMG	3.0	MOG2	2.875
GMG	4.125	**HSMD**	**3.875**	GSOC	4.125	KNN	3.5	KNN	3.625	MOG2	4.0	LSBP	4.25	GSOC	3.625	**HSMD**	**3.625**	CNT	3.375	LSBP	3.875
GSOC	4.5	KNN	4.25	**HSMD**	**4.5**	GMG	3.75	CNT	4.125	GSOC	4.25	MOG2	4.875	KNN	3.75	GMG	4.0	LSBP	4.875	KNN	4.375
MOG2	4.875	MOG	4.375	GMG	4.625	GSOC	5.5	MOG2	5.5	MOG	5.25	**HSMD**	**4.875**	CNT	4.75	GSOC	4.875	MOG2	5.125	CNT	4.375
MOG	5.0	GMG	5.5	KNN	4.875	LSBP	5.875	GMG	5.875	CNT	5.25	KNN	5.5	LSBP	5.25	KNN	5.875	MOG	5.25	GSOC	5.375
KNN	6.375	GSOC	6.0	MOG	6.625	MOG2	6.125	GSOC	6.25	GMG	5.5	GSOC	5.5	GMG	5.25	LSBP	5.875	KNN	5.5	MOG	6.0
CNT	6.75	LSBP	6.125	MOG2	6.75	MOG	6.625	**HSMD**	**6.5**	KNN	5.5	GMG	6.25	MOG2	6.75	CNT	6.0	**HSMD**	**6.625**	GMG	6.25

↓: the lowest result is the best.
The HSMD results are highlighted using dark grey for the HSMD, and the best results of other methods are highlighted using light grey.
R is the average ranking.
The baseline, cameraJitter, intObjMotion, shadow and thermal categories are shared between the CDnet2012 and CDnet2014 datasets. While the badWeather, dynamicBackground, nightVideos, PTZ, lowFrameRate and turbulence categories are specific to the CDnet2014 dataset.

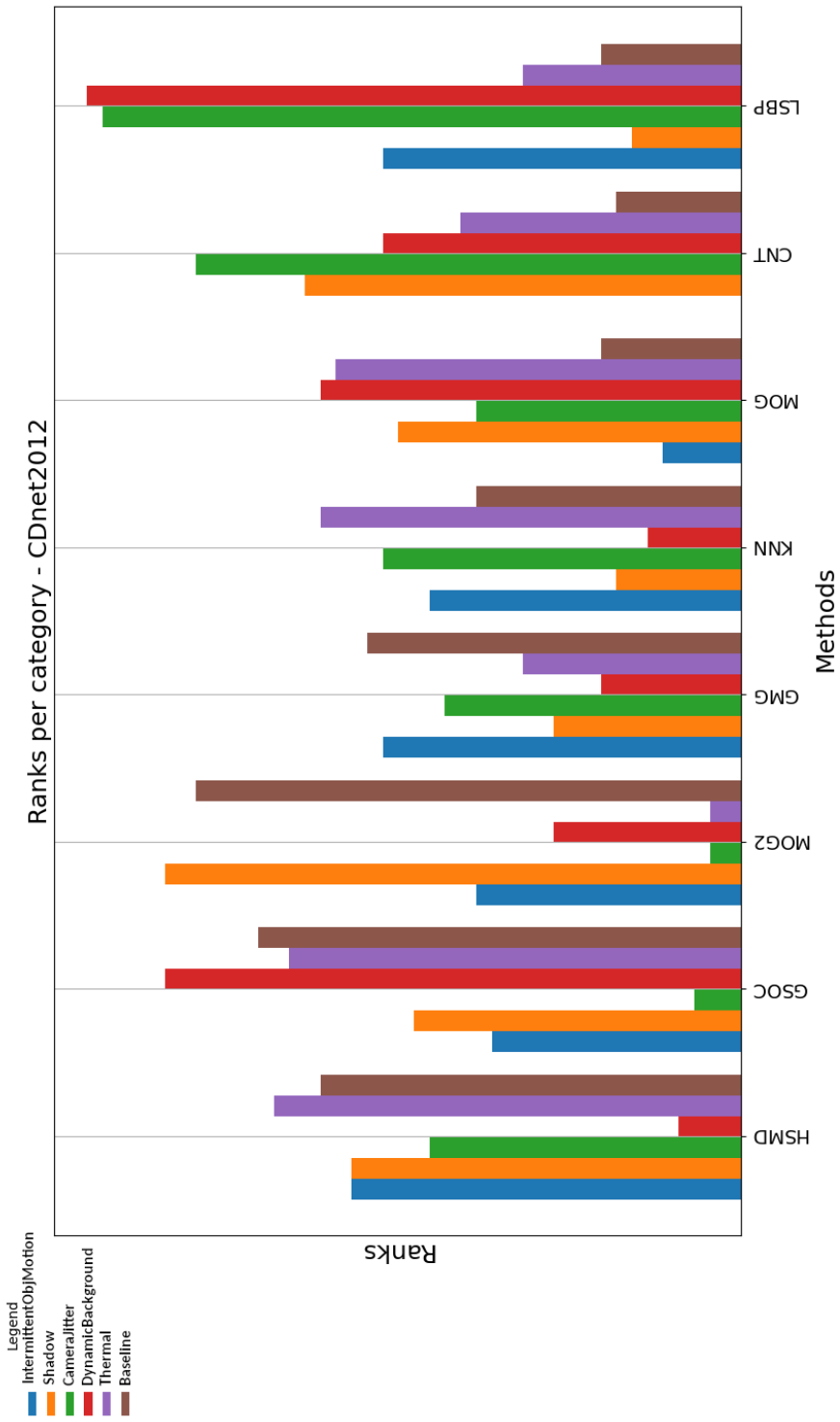

Figure 4.5: CDnet2012 overall result based on Average Ranking (R) per method. The highest bars show the higher ranks, and it is clear that none of the methods had the best ranks in all the categories. Furthermore, it is possible to see that the HSMD achieved high ranks across all the categories, except dynamic background.

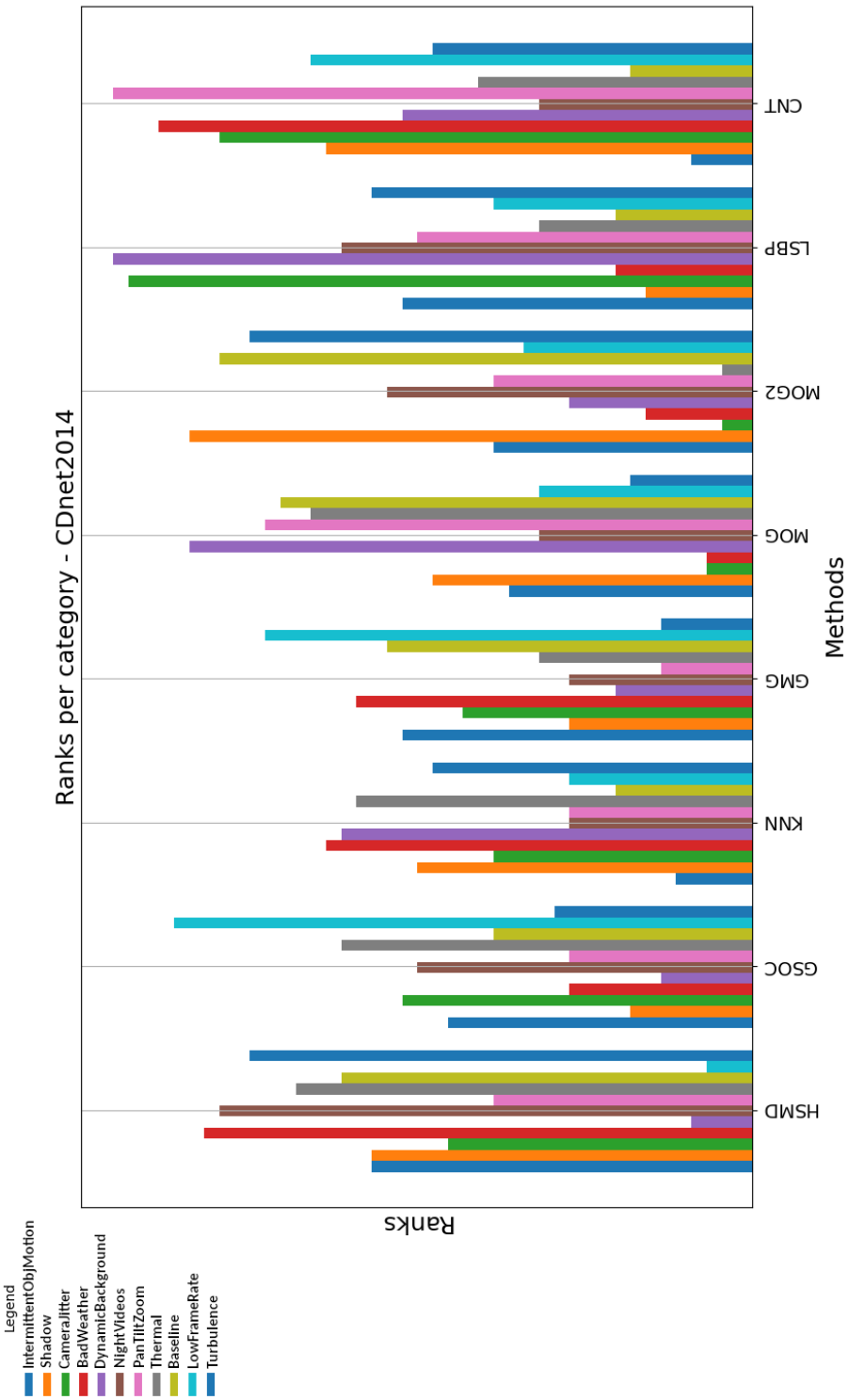

Figure 4.6: CDnet2014 overall result based on Average Ranking (R) per method. The highest bars show the higher ranks, and it is clear that none of the methods had the best ranks in all categories. Furthermore, it is possible to see that the HSMD achieved high ranks across most of the categories, except dynamic background and low frame rate.

Figures 4.5 and 4.6 show the variation of the ranks obtained per category and per method.

Based on the results shown in Table 4.5, Figure 4.5 and Figure 4.6, the HSMD performs better when processing image sequences from intermittent object motion, night vision, baseline, and turbulence categories. These categories contain moving objects with high contrast ration, which is ideal for sensing by the spiking neurons. Overall, the HSMD has improved the results of the GSOC in 8 of the 11 categories (see Table 4.5). It is also easy to infer that the HSMD exhibits the lowest R variation, which justifies the HSMD being ranked first.

4.4.3 Results analysis

The HSMD performed very poorly in the dynamic background and low frame rate categories, suggesting that the spiking neuron model is not ideal for distinguishing the type of motion. i.e. the spiking neurons detect motion but are unable to distinguish between a shadow or the object itself. This result is probably because, in vertebrate retinas, only the ganglion cells are spiking cells, suggesting that distinction between the main object and shadows is probably performed by other non-spiking cells. Nevertheless, the creation of the new approach incorporating both the GSOC algorithm and the

SNN, which emulates the basic OMS-GC functionality, clearly improves the accuracy of the GSOC algorithm.

The CDnet2012 and CDnet2014 datasets are composed of image files of different resolution, and accordingly, the processing times vary. The HSMD takes approximately 72.4ms (CDnet2014) and 71.9ms (CDnet2012) to process images of 720×480 on a 96-cores Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz equipped with 792 GB of DDR4 and 12.7 TB of disk space. The slight variations are related to other applications running in the background. Therefore, the HSMD is capable of processing images of 720×480 at an average speed of 13.82fps (CDnet2014) and 13.92fps (CDnet2012). Finally, the HSMD is the first hybrid SNN algorithm capable of processing images at such a frame rate, as far as the authors are aware.

4.5 Discussion

A bio-inspired HSMD has been proposed to detect object motion and assess against the CDnet2012 and CDnet2014 datasets. The HSMD was written in C++ using the C++ STL 17 (implementation of data structures), Boost 1.71 (file management), and OpenCV 4.5.0 (image processing). These incorporate video sequences of many moving objects under various challenging environmental con-

ditions and are widely used for benchmarking BS algorithms. The CDnet2012 is composed of 6 categories of movements, and the CDnet2014 augments the initial 6 to 11 categories of movements. Eight metrics, utilised as standard in the Change Detection datasets, were used to assess and compare the quality of the HSMD algorithm.

The HSMD performed poorly in the dynamic backgrounds and low frame rate categories, indicating that the spiking neuron model is not the best for classifying these types of object motion (i.e. the spiking neurons are able to detect motion but are unable to distinguish a shadow from the actual object). This deficiency is most likely caused by the fact that only the ganglion cells in vertebrate retinas are spiking cells, indicating that other non-spiking cells are most likely responsible for differentiating between the main object and shadows. However, the development of the new method that combines the GSOC algorithm with a customised SNN, which mimics the fundamental OMS-GC functionality, significantly increases the GSOC algorithm's accuracy. Nevertheless, the HSMD algorithm performed overall best in both the CDnet2012 and CDnet2014 while performing better than all the tested BS algorithms in the intermittent object motion, night videos, thermal and turbulence categories, second best in the bad weather category, and third-best in the base-

line and shadow categories. The comparatively good results are a consequence of using the SNN for emulating the basic functionality of OMS-GC, which improves the sensitivity of the HSMD to object motion. The HSMD is also the first hybrid SNN algorithm capable of processing video/image sequences with near real-time performance (i.e. 720×480@13.82fps [CDnet2014] and 720×480@13.92fps [CDnet2012]).

The main lesson learnt from the implementation of the HSMD was that the SNN latency need to be improved for targetting real-time applications. As already mentioned in Section 2.2, SNNs are massively parallel and can be accelerated using dedicated hardware such as FPGAs (see 2.6.3). The hardware implementation of the HSMD is covered in Chapter 5.

Chapter 5

NeuroHSMD: Neuromorphic Hybrid Spiking Motion Detection

The NeuroHSMD is the hardware implementation of the HSMD discussed in Chapter 4. As discussed in section 2.6.3, FPGAs offer the desired flexibility to accelerate massively parallel SNN architectures. Therefore, a high-end FPGA was selected for accelerating the HSMD's SNN. OpenCL was used to describe the customised SNN because it provides a higher level of abstraction than other HDL tools (see Section 5.2.4 for more details), increase in productivity, and compatibility with other compatible devices such as non-Intel FPGAs, CPUs and GPUs. The results show that the NeuroHSMD was 82% faster processing 720×480 than the HSMD algorithm. Furthermore, the NeuroHSMD was ranked first alongside the HSMD when benchmarked against CDnet2012 and CDnet2014, meaning

that there was no degradation in the quality of the foreground extraction.

5.1 Introduction

The human brain is characterised by its tolerance to faults/noise, concurrent processing capabilities, flexibility, and high level of parallelisation when processing data. Furthermore, the adult human brain has a power consumption of about 400 Kcal per day, equivalent to 25 Watts [311]. Again, the human brain can reach 10-50 petaflops, outperforming any COTS CPU [312]. Despite the fact that CPUs outperform the human brain by several orders of magnitude when processing and transmitting sequential signals, the human brain processes millions of signals in parallel using its massively parallel circuits [311; 312].

The human brain is composed of millions of interconnected neuron circuits composed of different types of neuron cells and contributing to specific brain computations [313; 314]. While CPUs transmit signals at a few tenths of a gigahertz, neuronal circuits transmit signals at hundreds of gigahertz [313; 314]. Nevertheless, the human brain can outperform CPUs when processing signals from complex systems such as the auditory and visual systems because of its massive parallel structure [314]

GPUs and FPGAs are parallel processing devices that can be used in conjunction with CPUs to accelerate parallellisable algorithms. GPUs are specialised electronic circuits with a flexible architecture designed for parallel processing of graphics and video rendering and accelerating some types of AI algorithms [315]. FPGAs are integrated circuits composed of built-in interconnected hardware blocks that can be freely reprogrammable after manufacturing [11]. In contrast to GPUs, which have a well-defined architecture, FPGAs are flexible devices that allow the user to describe new hardware architectures, such as brain-like, neuromorphic architectures.

The HSMD algorithm has proven to be very robust in extracting the foreground from the background (see section 4) as a direct consequence of using a SNN to emulate the basic functionality observed in OMS-GCs. Furthermore, the HSMD improves the GSOC algorithm by employing a customised SNN composed of three layers of interconnected neurons through a 1:1 synaptic connectivity. Nonetheless, the HSMD is substantially slower than the GSOC because the customised SNNs also introduced a substantial delay because they are, by their parallel nature, not optimised for sequential processing architectures such as CPU architectures. Therefore, FPGAs were the obvious choice for accelerating the HSMD's SNN because of their flexibility to describe massively parallel architectures. FPGAs

are typically reprogrammed using VHDL or Verilog, which are flexible but complex HDLs. In recent years, the FPGA manufacturers have invested in HLS tools to enable users to programme FPGAs using C-like programming languages. One of the most successful HLS tools is OpenCL, which allows users to program kernels that can be compiled targeting CPUs, GPUs, and FPGAs. The NeuroHSMD presented in this chapter was implemented using OpenCL on FPGAs.

The methodology is discussed in section 5.2; the results are presented in section 5.3 and the discussion of the NeuroHSMD results is performed in section 5.4.

5.2 Implementation details

SNNs are composed of a variable number of spiking neurons, and each neuron's output will contribute to the generation of spike events. Spiking neuron models are therefore highly parallelisable and not computationally suitable for CPUs. In this section, the details of the hardware implementation are discussed. Heterogeneous computing platforms are discussed in section 5.2.1; the architecture of FPGA devices is discussed in section 5.2.2; HDL are discussed in section 5.2.3, OpenCL framework is introduced in section 5.2.4; the selected hardware platform is discussed in section 5.2.5; and the de-

tails about the host application and its device kernels are described in section 5.2.6.

5.2.1 Heterogeneous computing platforms

The rise of AI and the continuous generation of big data are creating computational challenges. CPUs are not enough to efficiently run state-of-the-art AI algorithms or process all the data generated by a wide range of sensors. World-leading processing technology companies (such as NVIDIA, AMD, Intel and ARM) have been looking closely into the new requirements. They have been pushing the boundaries of technology to deliver efficient and flexible processing solutions.

Heterogeneous computing refers to the use of different types of processor systems in a given scientific computing challenge.

Heterogeneous platforms are composed of different types of computational units and technologies. Such media can be composed of multicore CPUs, GPUs and FPGAs acting as computational units and offering the flexibility and adaptability demanded by a wide range of application domains [316]. These computational units can significantly increase the overall system performance and reduce power consumption by parallelising concurrent operations that require substantial CPU resources over long periods.

Accelerators like GPUs and FPGAs are massive parallel processing systems that enable accelerating portions of code that are parallelisable. Combining CPUs with GPUs and FPGAs helps improve the performance by assigning different computational tasks to specialised processing systems. GPUs are optimised to perform matrix multiplications in parallel, which is the major bottleneck in video processing and computer graphics. Nevertheless, GPUs also introduce hardware and environmental limitations (e.g. high-power consumption and architectural limitations) [315]. SNNs are massively parallel in their nature and not suitable for matrix representation because each neuron can be considered a node containing several sequential mathematics operations. Therefore, a FPGA device was selected to accelerate the HSMD's customised SNN.

OpenCL is a C/C++-based programming language specially designed for software developers to write applications targeting heterogeneous computing platforms such as CPUs, GPUs, and FPGAs [34]. OpenCL provides an abstraction layer allowing compatibility across devices and enabling the same source code to run on different device architectures. Furthermore, OpenCL provides facilities for developers to control parallelism and make effective use of the target device resources [33]. The OpenCL framework was chosen to ensure that the NeuroHSMD is widely compatible with a wide

range of hardware devices (for more information on OpenCL, see section 5.2.4).

5.2.2 FPGA Architecture

FPGAs have been used, for many decades, accelerating applications, including edge/cloud computing. FPGAs are flexible devices because of their flexible architecture, enabling developers to describe customised architectures. Such flexibility comes with a downside because FPGAs are also known by their associated complexity. There are two main FPGA manufacturers, namely, Intel[1] and AMD-Xilinx[2]. The Computational Neurosciences and Cognitive Robotics (CNCR) group, where the PhD programme was developed, has different Intel FPGA development boards. The target board is fitted with a state-of-the-art Stratix 10 SoC FPGA device[3] (see further details in the next section 5.2.5). Therefore, this chapter focus on the Intel Stratix architecture.

The Intel Stratix family is composed of logic array blocks (LAB) made up of 10 basic building blocks called ALMs. Each ALM is composed of fractionable Look-Up-Tables, also known as adaptive

[1]Available online, https://www.intel.co.uk/content/www/uk/en/products/programmable/fpga.html, last accessed: 04/03/2021

[2]Available online, https://www.xilinx.com/, last accessed: 04/03/2021

[3]Available online, https://www.intel.co.uk/content/www/uk/en/products/programmable/soc/stratix-10.html, last accessed: 07/04/2021

look-up-table (ALU), two-bit full adder and four registers (see Figure 5.1).

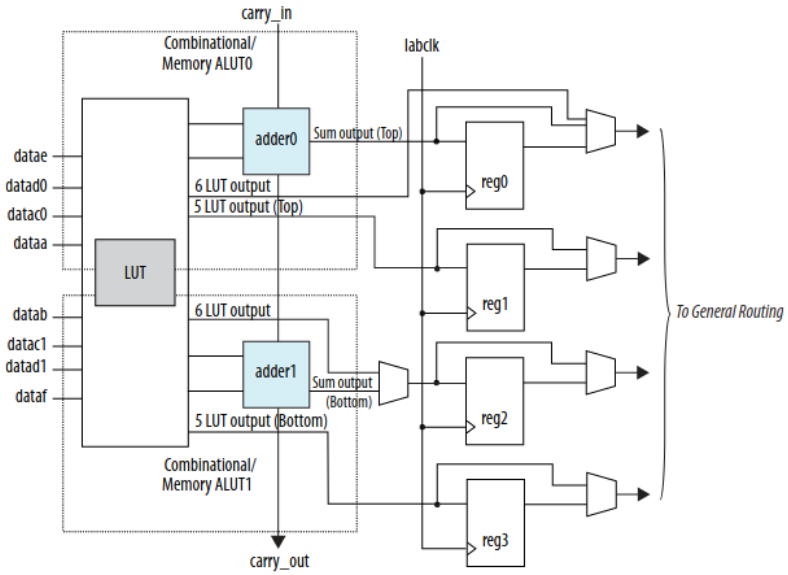

Figure 5.1: ALM Block Diagram. Each register has the following ports: i) Data in, ii) Data out, iii) Clock, iv) clock enable, v) synchronous clear and vi) asynchronous clear. Adopted from [10].

LABs can be freely reconfigured to implement logic and arithmetic functions. Furthermore, up to a quarter of the LABs can be used as memory logic array blocks (MLAB). Each LAB contains dedicated logic elements used to drive control signals to ALMs. Each MLAB supports up to 640 bits of simple dual-port SRAM. It is possible to configure each ALM in an MLAB as 32×2 memory blocks equivalent to $32 \times 2 \times 10$ simple dual-port SRAM blocks. Dual-port SRAMs are low-latency memory devices that only take a clock cycle to perform a read/write operation (for example, Syn-

chronous Dynamic Random Access Memory (SDRAM) in CPUs takes thousands of clock cycles to complete read/write operations).

5.2.3 Hardware Description Language

The behaviour of LABs and ALMs can be programmed using HDL. Although many HDLs being available, the Institute of Electrical and Electronics Engineers (IEEE) endorses the VHDL and Verilog. Although the VHDL syntax is identical to Pascal and the Verilog syntax is identical to C, both languages are easy to learn and hard to master.

HDLs are potent tools because they enable users to programme at the Register Transfer Level (RTL) (lowest level of coding), which is challenging to master. Hardware developers must have a deep knowledge of the reprogrammed device. Projects designed for a specific FPGA device might not work in another, even if they belong to the same FPGA family. Furthermore, the debugging of HDL source code is very slow and prone to errors, making applications a long and time-consuming process due to the complex FPGA architecture (see Figure 5.2).

FPGA manufacturers have been simplifying the FPGA design flow through the HLS. HLS tools deliver C/C++ like tools providing higher-level hardware abstraction, enabling software developers

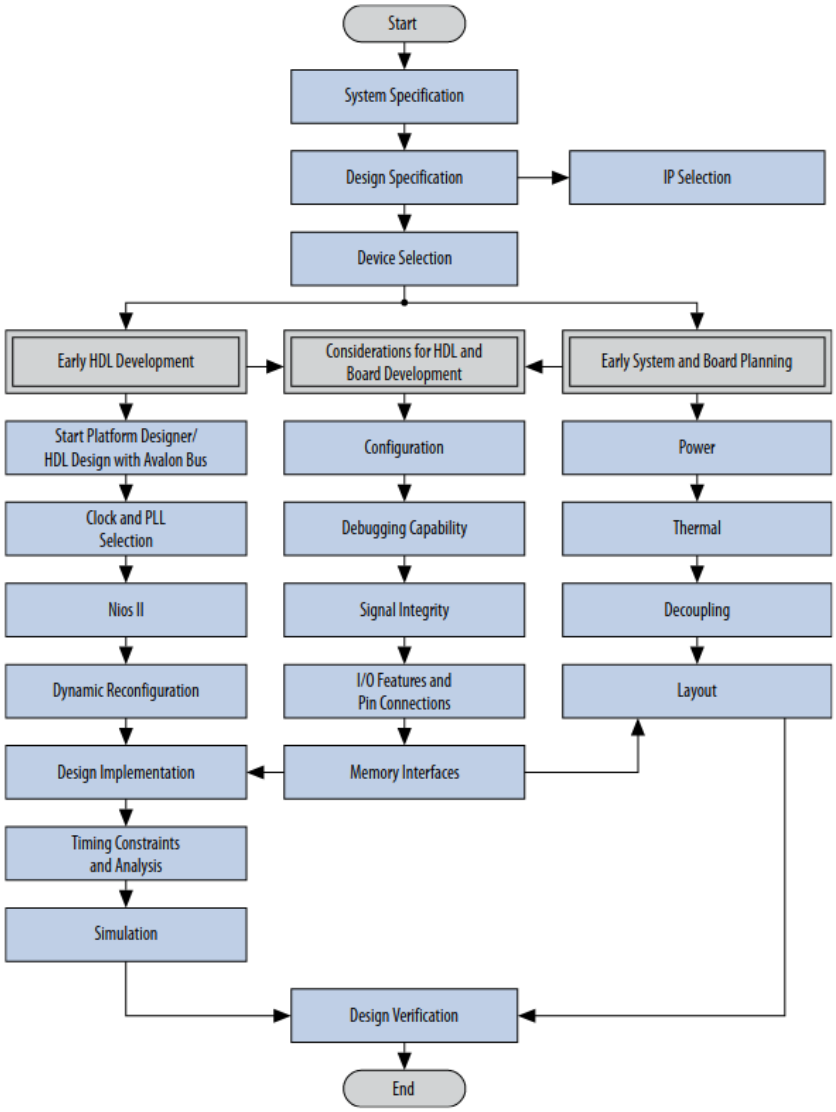

Figure 5.2: Intel Stratix 10 device design flow [11]. The main stages of the design flow include the system specification, device selection, early system and board planning, pin connection considerations for board design, I/O and Clock planning, design entry, design implementation, analysis and optimisation and verification. Adopted from [12].

to use FPGAs. OpenCL, a HLS framework, to be discussed in the next section 5.2.4 was used to avoid having to use HDL when implementing the NeuroHSMD. Moreover, the NeuroHSMD's SNN was

written in C++ for OpenCL and automatically converted to HDL using Intel FPGA tools.

5.2.4 OpenCL

OpenCL applications are split into two parts, namely, **host** application(s) and **device** kernel(s) (see Figure 5.3). The **host** application(s) is(are) always compiled on the host Operating System and run on a CPU. **Host** applications are also used to launch the target kernels on the target **devices**. Kernels are special functions written in OpenCL C/C++ to perform parallelisable computations on accelerators such as GPUs and FPGAs [33]. For instance, consider two m×n matrices, A and B where it is expected to do the operation C=A+B where C is the third matrix of m×n. In this case, the kernel could just perform, in parallel, the addition of matrices A and B elements and store the result in C. Unlike in CPUs, where it would take m×n to do this matrix addition, GPUs and FPGAs could parallelise this operation depending on the resources available per device resulting in the acceleration of the application.

Buffer objects within a context are used in OpenCL to exchange data between the host and device [317]. The Intel FPGA SDK for OpenCL offline compiler optimises the kernel throughput by adjusting buffer sizes during the kernel compilation process [28].

OpenCL provides both mapped and asynchronous buffers, enabling the application to continue to run while additional data is exchanged (see Figure 5.3).

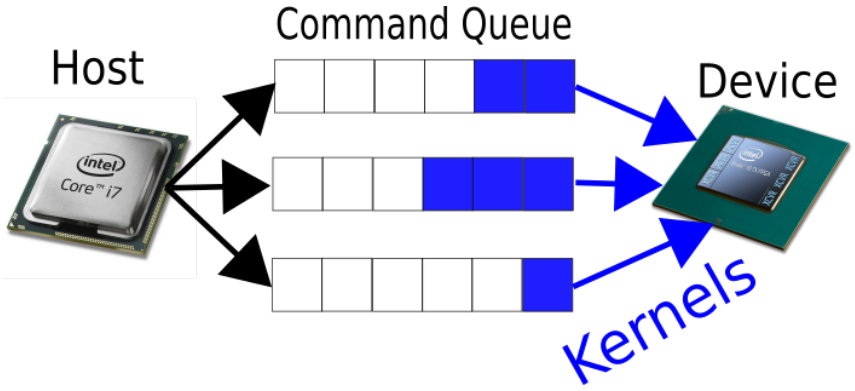

Figure 5.3: Representation of a OpenCL host application and three device kernels

Software developers have to carefully analyse the code to be optimised and only select the sections that may benefit from hardware acceleration because the maximum speed is always dictated by the PCIe bus speed. Another big challenge for software developers is the low debugging capabilities available while the code is being executed on the device.

Although it is possible to use OpenCL to program FPGA and GPU devices, GPUs are specialised devices designed for video rendering and graphics processing. At the same time, FPGAs are customisable devices that can be freely reconfigurable. Therefore, FPGAs offer more flexibility than GPUs, which is desirable for accelerating SNNs because they can be modelled using the Network-on-

Chips (NoCs) concept. Each individual spiking neuron in the node interconnects to one or more nodes of the same SNN. The flexibility offered by both FPGAs and OpenCL makes selection of FPGAs over GPUs the obvious choice.

Intel FPGA SDK for OpenCL (IOCL) provides a compiler and powerful tools to build and run OpenCL applications targeting Intel FPGA devices. The IOCL generates two main components: the host application and the FPGA programming bitstream(s). The IOCL offline compiler (AOC) first compiles the custom kernel(s) to an image file (*.aocx) that will be used to program the FPGA. In contrast, the host-side C/C++ compiler compiles the host application and then links it to the IOCL runtime libraries (see Figure 5.4).

The DE10-pro platform provides the board support package (*.bsp). The AOC targets the DE10-pro platform when compiling an OpenCL kernel to generate the ***.aocx** object that is **only compatible** with the DE10-pro's Intel Stratix 10 FPGA device. The IOCL utility programme (aocl) is used to programme the FPGA device using the image file ***.aocx** is then used to programme the FPGA to enable the host application to exchange data with the kernel using OpenCL buffers via the PCIe bus. Multiple FPGAs can be used by the same host application (see Figure 5.5).

The IOCL compiles one or more OpenCL kernels and creates a

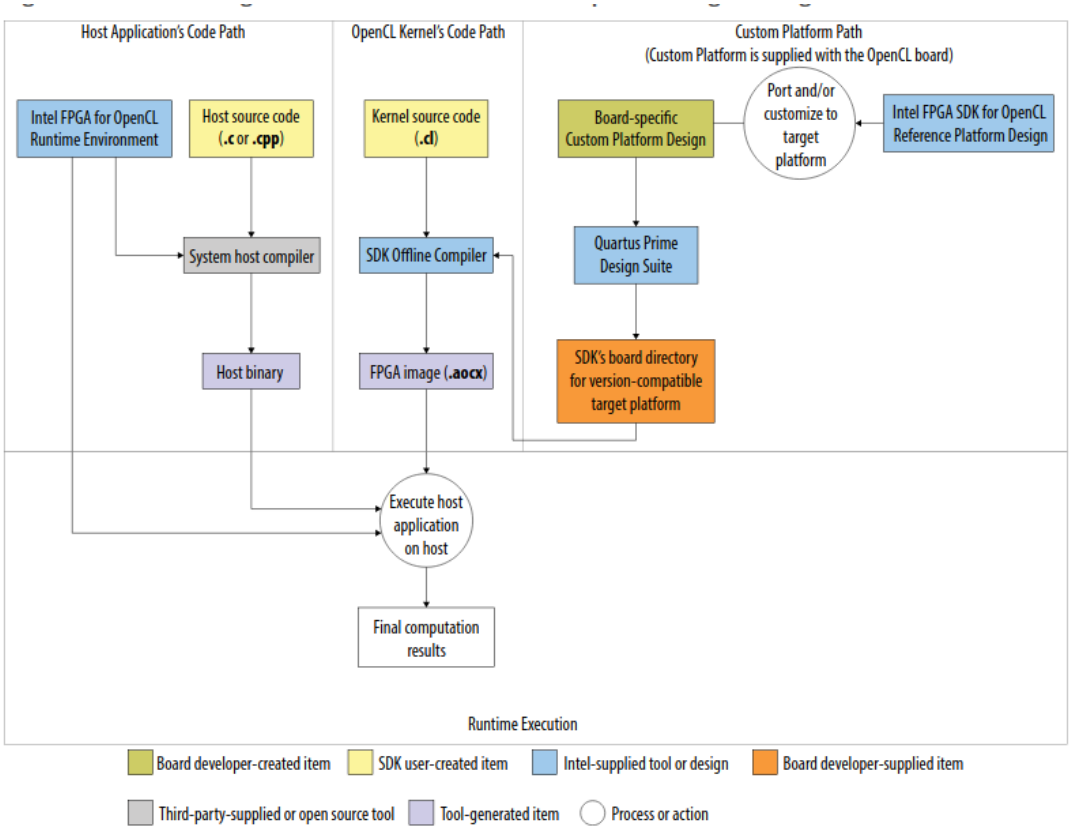

Figure 5.4: Intel FPGA SDK for OpenCL FPGA design flow. Adopted from [13].

hardware configuration file. After a successful compilation, the files ***.aocr**, ***.aoco**, (*.aocx), and reports/report.html are generated (see Figure 5.6). The report.html contains the estimated resource usage and a preliminary assessment of area usage. The intermediary ***.aoco** and ***.aocr** are only used in the generation of the ***.aocx** which is then used to programme the FPGA.

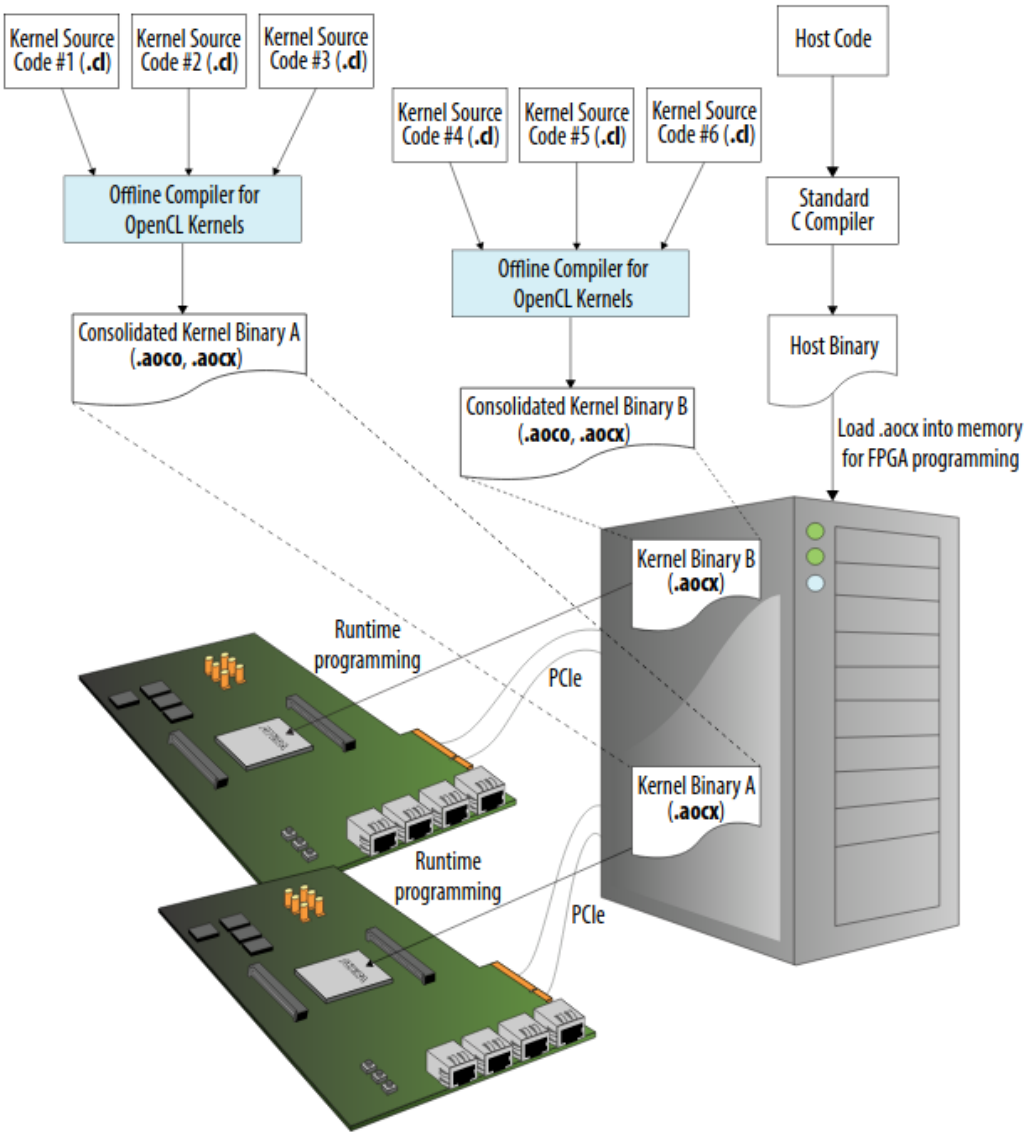

Figure 5.5: Intel FPGA SDK for OpenCL FPGA programming flow. Adopted from [13].

5.2.5 Hardware platform

Although there are hundreds of FPGA boards, only a few boards provide board support packages (BSP) for OpenCL. The creation

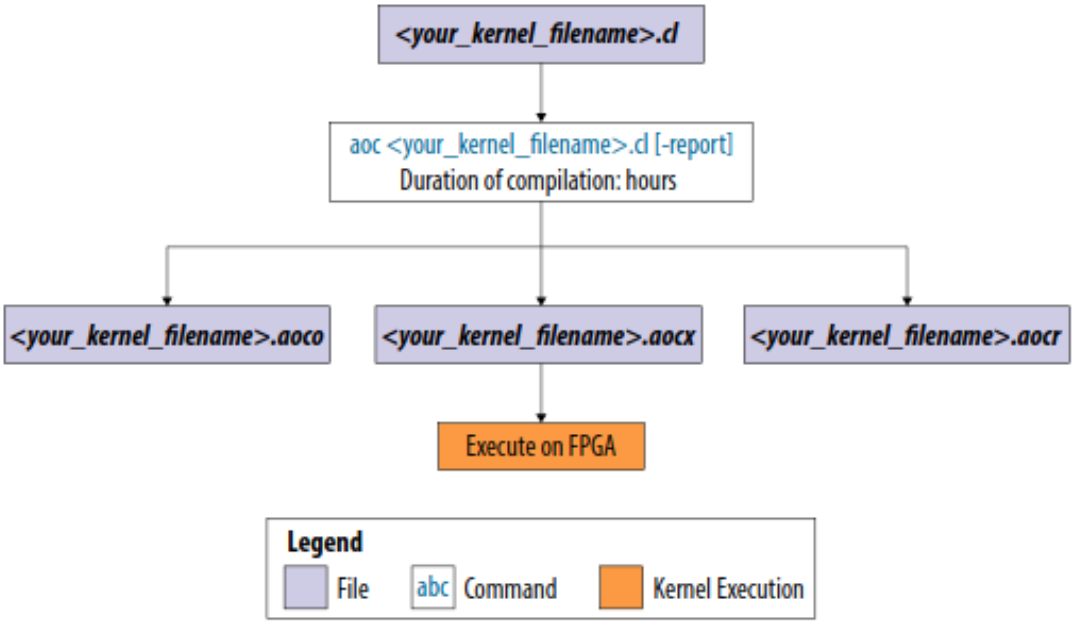

Figure 5.6: OpenCL compilation flow. Adopted from [13].

of the OpenCL BSP for a given FPGA is a complex and time-consuming process that considers the FPGA device and how the Inputs/Outputs are routed on the physical board. Therefore, the selection of the FPGA board should be made taking into consideration the size of the FPGA device, the OpenCL BSP, the version of the BSP to ensure compatibility with recent Linux distributions and the reference/user manuals.

The Terasic DE10-pro development board [14] (see Figure 5.7) was used for implementing the NeuroHSMD discussed in this chapter. The Terasic DE10-pro development board is equipped with a state-of-the-art high-end Intel Stratix 10 FPGA device. According

to Terasic, the DE10 pro was created to meet the needs of AI, data centers, and high-performance computing. Furthermore, the DE10-pro development board takes advantage of the latest Intel Stratix 10 to obtain high-speed and low-power (with up to 70% lower power). It is equipped with a 32GB DDR4 memory module running at over 150 Gbps, up to 15.754 GB/s data transfer via PCIe Gen 3 x16 edge between FPGA and the host PC, and 4 onboard QSFP28 (100GbE) connectors.

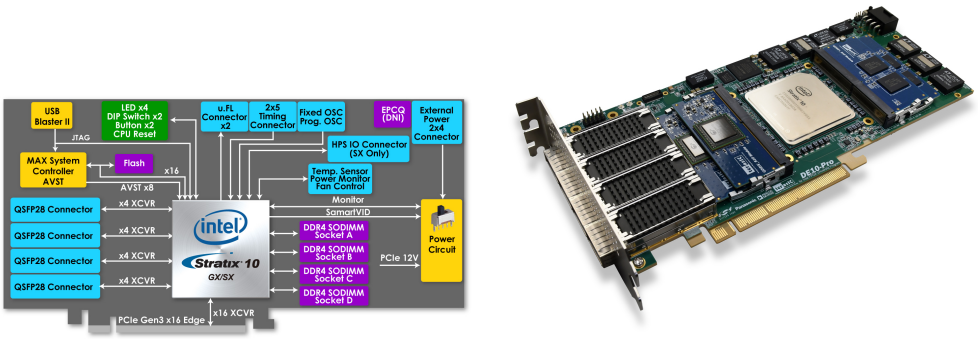

Figure 5.7: Terasic DE10 pro development kit. (left) block diagram and (right) DE10 pro board. Adopted from [14]

The DE10 pro was installed on the host PC equipped with an Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz and 16 GB of DDR3 using the PCIe slot.

5.2.6 NeuroHSMD implementation

The NeuroHSMD's SNN was written in C++ for OpenCL. In OpenCL, the data flow between host application and FPGA kernel is as fol-

lows: a) allocate and specify buffer types (read/write) on the host and device; b) copy FPGA data from application data structures to host buffers; c) transfer data from host buffers to device buffers; d) run the inference on the device; e) copy the results from the device to host buffers; f) copy data from host buffers to application data structures (see Figure 5.8).

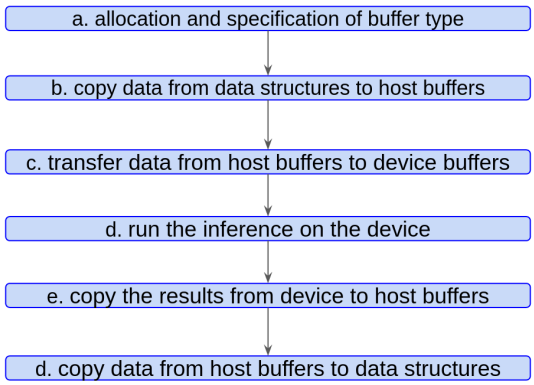

Figure 5.8: OpenCL computation stages. The stages include: a) allocation and specification of buffer types on the host and device; b) copying data from the application data structures to host buffers; c) transferring data from host buffers to device buffers; d) running the inference on the device; e) copying the results from the device to host buffers; f) copying data from the host buffers to application data structures.

The NeuroHSMD algorithm performs the following computation stages: 1) image capture, 2) conversion from colour to grey scale, and 3) background subtraction using the OpenCL's GSOC algorithm, buffer the results and transfer buffered results to the FPGA device; 4) run the FPGA inference and wait for the spike results;

5) run the SNN kernel; 6) apply average filter; and 7) display and save the output image.

The NeuroHSMD's processing stages are summarised in the diagram 5.9 and the NeuroHSMD architecture is depicted in Figure 5.10.

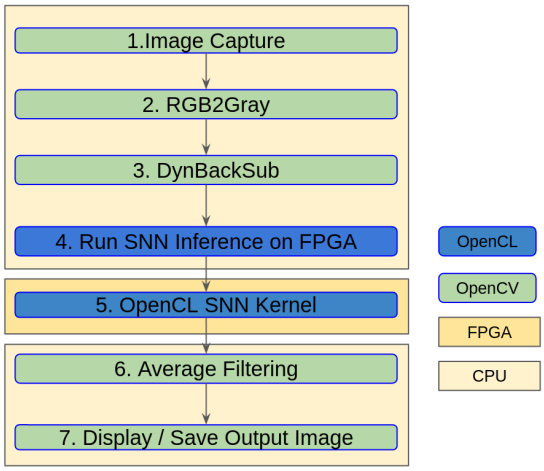

Figure 5.9: NeuroHSMD computation stages. The OpenCL implementation is represented in blue and the OpenCV in green. The light yellow background represents the computation stages that run on the CPU (i.e. steps 1, to 4, and 6 to 7), and in light orange, the stage that runs on the FPGA device (i.e. step 5)

Furthermore, the NeuroHSMD implementation can be split into the NeuronHSMD Host Application (NHA) and the NeuronHSMD device kernels (NDK). The NHA and NDK are described in detail in the sections 5.2.6.1 and 5.2.6.2, respectively.

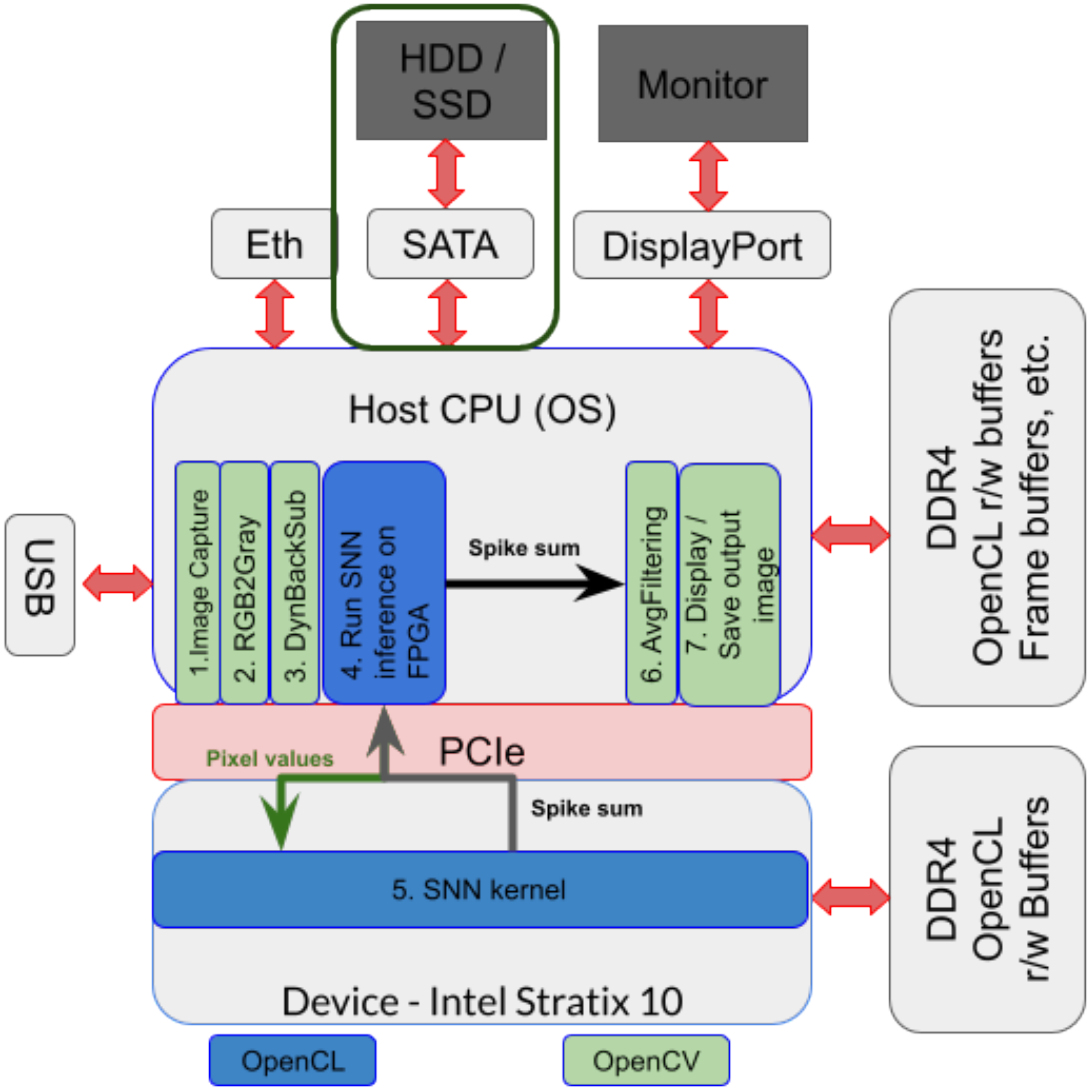

Figure 5.10: NeuroHSMD architecture. The diagram represents the computation stages that run both on the CPU and FPGA. Shows that the FPGA is connected to the host CPU via the PCIe bus. It also shows the dedicated memory of the CPU and FPGA device. This also includes how external devices (e.g. image sensors, monitor, and HDD/SSD) connect to the host CPU via different interfaces (e.g. Ethernet (eth), SATA, display port, and usUSBb). The computation stages implemented in OpenCL are in blue and OpenCV in green.

5.2.6.1 Host application

The NHA is used to interface the two NDK. The NHA performs the first stages of pre-processing and performs inference of the SNN kernel (described on the FPGA), and uses the inference results to compute the BS . Furthermore, the NHA can process both live images captured from camera devices or extracted from videos stored on USB or SATA devices. The NHA is used to interface the NDKs. The NHA performs the first stages of pre-processing and performs inference of the NDK (described on the FPGA), and uses the inference results to compute the BS. Furthermore, the NHA can process both live images captured from camera devices or extracted from videos stored on USB or SATA devices.

Algorithm 3 summarises each of the computation stages that occur in the NHA.

The communication between the NHA and the NDK is limited by the PCIe bus speed (16 GB/s [1]). In the HSMD, the limitations are only dictated by the CPU speed and the DDR4 memory speed (34.1 GB/s[2]) which two times faster than the PCIe bus speed.

[1] Available online, https://www.trentonsystems.com/blog/pcie-gen4-vs-gen3-slots-speeds, last accessed: 21/06/2021

[2] Available online, https://www.crucial.com/support/articles-faq-memory/understanding-cpu-limitations-with-memory, last accessed: 21/06/2021

Algorithm 3 NHA

Input:
img: image frame;
Output:
post_proc_img: post processed image;
stats: computation statistics;
Main Algorithm:

```
1: initialise_opencl()
2: for iterator ← list_folders.begin() to list_folders.end() do
3:    (x, y) ← get_image_size();
4:    num_layers ← 3
5:    tot_neurons ← x.y.num_layers
6:    reset_opencl_buffers(tot_neurons )
7:    gsoc ← initialise_gsoc_back_subtraction
8:    for iterator2 ← files_list.begin() to files_list.end() do
9:       img ← read_image(iterator2);
10:      < pixel_values >← gsoc.compute(img)
11:      aocl_snn_v1(< pixel_values >) →< spike_sum > {Alg. 4}
12:      OR
13:      aocl_snn_v2(< pixel_values >) →< spike_sum > {Alg. 5}
14:      post_proc_img ← get_spikes_sum_l3(< spike_sum >)
15:      save(post_proc_img)
16:      stats ← compute_stats(time)
17:   end for
18:   save(stats)
19: end for
```

5.2.6.2 Device kernels

The NDKs section covers the two kernels (i.e. NeuroHSMDv1 and NeuroHSMDv2) that have been implemented. The only difference between the two NDKs is that the *aocl_snn_v2* only computes the spike sums for neurons that have the pixel intensity values above 0.0.

The *aocl_snn_v1* was parallelised by a factor of 16. The NDK version 1 is the equivalent implementation of the HSMD algorithm described in chapter 4 which is referred to as HSMDv1 in this chapter.

The algorithm 5, inferred by the NHP, summarises the computation steps required to compute the spike sum.

The *aocl_snn_v2* was parallelised by factor of 16. The NDK version 2 contains an optimisation where the spike sum for a given neuron of layer 1 is only computed if the pixel intensity value is greater than 0.0. This way, the computations will only be done for neurons that receive stimuli from pixels that belong to the foreground.

This optimisation was also applied to the original HSMD algorithm described in chapter 4. In this chapter, the optimised version of HSMD (i.e. where the spike sum for a given neuron of layer 1 is only computed if the pixel intensity value is greater than 0.0) is referred to as HSMDv2.

Algorithm 4 aocl_snn_v1

Parallel Circuits: 16
Constants:
R_m: membrane resistance;
τ_m: membrane time constant;
dt: time step;
$p2c$: pixel values to current;
$steps$: number of steps;
$s2c$: spike to current;
Input:
$< pixel\_val >$: pixel values;
num_neuron_layer: number of neurons per layer;
Output:
$< spk\_sum >$: spike sum;
Main Algorithm:

```
1: for neuron_idx ← 0 to number_neurons do
2:    I_s ← pixel_val[neuron_idx].p2c;
3:    for dt1 ← 0 to steps do
4:       Layer 1:
5:       compute_V_m_l1[neuron_idx](I_s);
6:       update_spike_sum_l1[neuron_idx](V_m_l1);
7:       Layer 2:
8:       I_s_l2 ← spike_sum_l1[[neuron_idx].s2c;
9:       compute_V_m_l2[[neuron_idx](I_s_l2);
10:      update_spike_sum_l2[neuron_idx](V_m_l2);
11:      Layer 3:
12:      I_s_l3 ← spike_sum_l2[neuron_idx].s2c;
13:      compute_V_m_l3[neuron_idx](I_s_l2 + I_s_l3);
14:      update_spk_sum[neuron_idx](V_m_l3);
15:   end for
16: end for
```

Algorithm 5 aocl_snn_v2

Parallel Circuits: 16

Constants:

R_m: membrane resistance;
τ_m: membrane time constant;
dt: time step;
$p2c$: pixel values to current;
$steps$: number of steps;
$s2c$: spike to current;

Input:

$< pixel\_val >$: pixel values;
num_neuron_layer: number of neurons per layer;

Output:

$< spk\_sum >$: spike sum;

Main Algorithm:

```
1: for neuron_idx ← 0 to number_neurons do
2:   if pixel_val[neuron_idx] > 0.0 then
3:     I_s ← pixel_val[neuron_idx].p2c;
4:     for dt1 ← 0 to steps do
5:       Layer 1:
6:       compute_V_m_l1[neuron_idx](I_s);
7:       update_spike_sum_l1[neuron_idx](V_m_l1);
8:       Layer 2:
9:       I_s_l2 ← spike_sum_l1[[neuron_idx].s2c;
10:      compute_V_m_l2[[neuron_idx](I_s_l2);
11:      update_spike_sum_l2[neuron_idx](V_m_l2);
12:      Layer 3:
13:      I_s_l3 ← spike_sum_l2[neuron_idx].s2c;
14:      compute_V_m_l3[neuron_idx](I_s_l2 + I_s_l3);
15:      update_spk_sum[neuron_idx](V_m_l3);
16:    end for
17:  end if
18: end for
```

5.2.7 Datasets and benchmark

The NeuroHSMDv1, NeuroHSMDv2, HSMDv1 and HSMDv2 were tested against the CDnet2012 [9] and CDnet2014 [32]. The scripts provided by Nil Goyette et al. [9] were used with the same protocol used to test the HSMDv1 algorithm reported in chapter 4. The scripts provided by Nil Goyette et al. [9] enables the computation of the eight metrics (Re, Sp, FPR, FNR, PWC, PCC, Pr and F1) and rank the results (RC). More details about the eight metrics and the ranks' procedure can be found in section 4.3.2.

The benchmark of the four algorithms is required to ensure that the four algorithms produce comparative results to that of the original HSMDv1 results when tested against CDnet2012 and CDnet2014 datasets. Furthermore, the OpenCL calls the Intel Quartus, which performs several hardware optimisations that may include converting from floating-point to fixed-point representation, which might affect the accuracy of the NeuroHSMD algorithms during the synthesis step (one of the steps of the OpenCL design flow).

5.3 Results

The HSMDv1, HSMDv2, NeuroHSMDv1 and NeuroHSMDv2 were all tested on the same computer equipped with a quad-core Intel(R)

Core(TM) i7-4770 CPU @ 3.40GHz, 16GB of DDR3 @ 1600 MHz and 1TB of HDD. The results section is divided into three subsections. Namely, section 5.3.1 shows the resources' usage to enable the comparison between the two kernels' complexity, the speed performance results are presented in sections 5.3.2 and section 5.3.3 shows the benchmark results when tested against the CDnet2012 and CDnet2014 datasets.

5.3.1 Resources Usage

The resources' usage are given the report generated by the *aoc* after the successful completion of the kernel compilation, which can take several hours (typically between 6h and 24h depending on the kernel complexity for the DE10pro). The resources' usage for the compilation of the NeuroHSMDv1 kernels is given in table 5.1 and the NeuroHSMDv2 kernel in table 5.2.

Table 5.1 NeuroHSMDv1 resources usage

Summary					
Info					
Project Name	snn_pc_v1				
Target Family, Device, Board	Stratix 10, 1SG280LU3F50E1VGS1, de10_pro:s10_sh2e1_4Gx2				
AOC Version	19.1.0 Build 240				
Quartus Version	19.1.0 Build 240 Pro				
Command	aoc device/snn_pci_v1.cl -o bin_acl/snn_pci_v1.aocx -v -report -board=s10_sh2e1_4Gx2 -incremental				
Quartus Fit Clock Summary					
Frequency (MHz)	306.25 (fmax)				
Quartus Fit Resource Utilisation Summary					
	ALMs	**FFs**	**RAMs**	**digital signal processings (DSPs)**	**MLABs**
Full design (all kernels)	387168.7	985558	2231	1408	4766
snn	488257.1	1060084	2484	1024	3871
Kernel Summary					
Kernel Name	Kernel Type	Autorun	Workgroup Size	# Compute Units	Hyper-Optimised Handshaking
snn	488257.1	1060084	2484	1024	3871
Estimated Resource Usage					
Kernel Name	**ALUs**	**FFs**	**RAMs**	**DSPs**	**MLABs**
snn	488257.1	1060084	2484	1024	3871
Global Interconnect	10629	16485	61	0	0
Board Interface	13132	20030	112	0	0
System description ROM	2	71	2	0	0
Total	567523 (30%)	907449 (24%)	2963 (25%)	976 (17%)	3786
Available	1866240	3732480	11721	5760	0
Compile Warnings					
None					

From the analysis of table 5.1 can be seen that the estimated resource utilisation is more pessimistic than the final resources' utilisation, which is a direct consequence of the Intel Quartus' optimisations during the synthesis and routing phases and to ensure that the circuit fits in the FPGA device. Nevertheless, it takes about 5 minutes to get the *estimated resources usage and between 6 and 24 hours to get the resource usage summary.* Therefore, it is a good practise to define the coefficient **N** in the statement # **PRAGMA UNROLL N** based on the *estimated resources' usage.*

Table 5.2 NeuroHSMDv2 resources usage

Summary					
Info					
Project Name	snn_pci_v2				
Target Family, Device, Board	Stratix 10, 1SG280LU3F50E1VGS1, de10_pro:s10_sh2e1_4Gx2				
AOC Version	19.1.0 Build 240				
Quartus Version	19.1.0 Build 240 Pro				
Command	aoc device/snn_pci_v2.cl -o bin_acl/snn_pci_v2.aocx -board=s10_sh2e1_4Gx2 -v -report -incremental				
Quartus Fit Clock Summary					
Frequency (MHz)	300 (Kernel fmax)				
Quartus Fit Resource Utilisation Summary					
	ALMs	**FFs**	**RAMs**	**DSPs**	**MLABs**
snn	486808.4	1034661	2486	1024	3933
Kernel Summary					
Kernel Name	Kernel Type	Autorun	Workgroup Size	# Compute Units	Hyper-Optimised Handshaking
snn	Single work-item	No	1,1,1	1	Off
Estimated Resource Usage					
Kernel Name	**ALUs**	**FFs**	**RAMs**	**DSPs**	**MLABs**
snn	530436	844793	2868	976	3844
Global Interconnect	10629	16485	61	0	0
Board Interface	13132	20030	112	0	0
System description ROM	2	71	2	0	0
Total	554199 (30%)	881379 (24%)	3043 (26%)	976 (17%)	3844
Available	1866240	3732480	11721	5760	0
Compile Warnings					
None					

Again, from the analysis of table 5.2 it can be seen that the estimated resource utilisation is higher than the actual resource utilisation as a consequence of the Intel Quartus's optimisations during the synthesis and routing phases.

The NDK v2 consumes 1448.7 ALMs less, 25423 FFs less, 2 RAMs less and the same number of DSPs as the NDK v1. Nevertheless, the NDK v2 kernel max frequency is 300MHz, while the NDK v1 kernel max frequency is 306.25 MHz. The NDK v2 enables the saving of less than 1% of resources and introduces a 2% increase in latency.

To ensure the best use of FPGA resources, the coefficient N should always be a multiple of $2n$. For example, the resources' usage of $N = 48$ is equivalent to $N = 64$. Moreover, both NDKs had failed to compile when $N = 32$ because there was not enough ALUs. The design required more ALUs than those available on the device, violating the compilation rules because the design would not fit on the device.

5.3.2 Speed performance

Table 5.3 displays the speed results obtained for the four algorithms tested against the CDnet2012.

Table 5.3 CDnet2012 speed result

Category	no. imgs	height	width	NeuroHSMDv2	NeuroHSMDv1	HSMDv2	HSMDv1
baseline/PETS2006	1199	576	720	23.66	20.45	8.40	9.73
cameraJitter/badminton	1149	480	720	28.33	25.48	10.01	11.50
dynamicBackground/fall	3999	480	720	27.28	25.01	9.87	11.08
shadow/copyMachine	3399	480	720	28.56	25.86	10.04	10.98
dynamicBackground/fountain01	1183	288	432	55.17	58.30	27.40	29.99
dynamicBackground/fountain02	1498	288	432	56.12	58.13	27.64	30.37
intermittentObjectMotion/abandonedBox	4499	288	432	55.93	58.45	26.88	29.89
intermittentObjectMotion/tramstop	3199	288	432	55.35	58.66	27.05	29.59
thermal/park	599	288	352	55.72	59.96	28.99	33.02
shadow/peopleInShade	1198	244	380	66.85	74.06	36.05	38.74
baseline/highway	1699	240	320	72.48	85.70	46.97	53.00
baseline/office	2049	240	360	69.49	78.86	40.24	46.71
baseline/pedestrians	1098	240	360	69.59	78.73	40.18	47.00
cameraJitter/boulevard	2499	240	352	68.96	78.81	41.21	46.98
cameraJitter/sidewalk	1199	240	352	69.04	78.54	39.49	47.32
cameraJitter/traffic	1569	240	320	72.17	85.29	43.95	51.40
dynamicBackground/boats	7998	240	320	71.86	84.29	45.33	51.70
dynamicBackground/canoe	1188	240	320	71.98	83.86	44.15	52.95
dynamicBackground/overpass	2999	240	320	71.98	84.56	43.69	49.31
intermittentObjectMotion/parking	2499	240	320	72.98	85.21	44.68	48.55
intermittentObjectMotion/sofa	2749	240	320	73.56	86.46	44.37	47.96
intermittentObjectMotion/streetLight	3199	240	320	72.01	84.95	44.31	47.76
intermittentObjectMotion/winterDriveway	2499	240	320	72.98	85.88	44.72	49.21
shadow/backdoor	1999	240	320	72.06	85.77	44.21	48.87
shadow/bungalows	1699	240	360	68.61	78.36	39.80	42.25
shadow/busStation	1249	240	360	68.79	78.78	39.85	42.91
shadow/cubicle	7399	240	352	70.87	80.47	41.07	44.52
thermal/corridor	5399	240	320	73.80	87.01	44.58	48.10
thermal/diningRoom	3699	240	320	73.38	86.60	44.38	47.82
thermal/lakeSide	6499	240	320	73.91	86.80	45.62	49.64
thermal/library	4899	240	320	74.83	87.05	44.40	49.36

Best results are highlighted using grey.

From Table 5.3 can be seen that both the NeuroHSMDv1 and NeuroHSMDv2 have performed better than the software versions (i.e. HSMDv1 and HSMDv2). It is also apparent that the NeuroHSMDv2 performs better in images with higher resolution (i.e. 720×480 and 720×576) while the NeuroHSMDv1 in lower resolutions (i.e below 720×480). Although the HSMDv2 is always faster than the HSMDv1 (non-opimised version), the NeuroHSMDv2 is only more efficient for resolutions above 288×432 and NeuroHSMDv2 is faster for lower size images. This is because FPGA optimisations

require the utilisation of more resources, which might increase latency.

Overall, the NeuroHSMDv1 had an average frame rate of 71.50 fps, NeuroHSMDv2 63.20 fps, HSMDv1 40.26 fps, HSMDv2 36.11 fps. Finally, the average frame rate for processing images with the native resolution of 720 × 480 per algorithm is i) NeuroHSMDv2 28.06 fps, NeuroHSMDv1 25.45 fps, HSMDv2 11.19 fps and HSMDv1 9.97 fps.

The speed results obtained for the four algorithms when tested against the CDnet2014 are depicted in table 5.4

Table 5.4 shows that the NeuroHSMDv1 and NeuroHSMDv2 have performed better than the software versions (i.e. HSMDv1 and HSMDv2) when tested against the CDnet2014 dataset. Once again, the NeuroHSMDv2 performs better in images with higher resolution (i.e. equal or higher than 480 × 295) while the NeuroHSMDv1 in lower resolutions (i.e below 480 × 295). Again, the NeuroHSMDv2 is only more efficient for resolutions above 288 × 432 because of the complexity and latency introduced by the optimisation circuit.

Overall, the NeuroHSMDv1 has an average frame rate of 43.51 fps, NeuroHSMDv2 39.94 fps, HSMDv2 29.30 fps, and HSMDv1 25.18 fps. It is essential to highlight that the CDnet2014 has more categories and image sequences, leading to different frame rates for

Table 5.4 CDnet2014 speed results

Category	no. imgs	height	width	NeuroHSMDv2	NeuroHSMDv1	HSMDv2	HSMDv1
badWeather/blizzard	6999	480	720	29.70	24.55	10.12	11.21
badWeather/snowFall	6499	480	720	29.61	24.31	10.16	11.11
badWeather/wetSnow	3499	540	720	26.54	21.76	8.93	10.19
baseline/PETS2006	1199	576	720	25.04	20.36	8.52	9.53
cameraJitter/badminton	1149	480	720	28.58	25.07	9.80	11.00
dynamicBackground/fall	3999	480	720	27.48	24.75	13.90	10.96
shadow/copyMachine	3399	480	720	28.76	25.66	14.10	11.51
turbulence/turbulence0	4999	480	720	28.59	25.07	14.26	11.62
turbulence/turbulence1	3999	480	720	28.24	25.21	14.28	11.35
turbulence/turbulence3	2199	486	720	28.72	25.15	14.07	11.49
PTZ/continuousPan	1699	480	704	28.49	23.91	9.88	10.85
lowFramerate/tunnelExit_0_35fps	3999	440	700	31.45	28.23	15.94	12.47
nightVideos/fluidHighway	1363	450	700	31.17	27.64	15.27	12.10
turbulence/turbulence2	4499	315	645	41.85	39.59	23.90	18.93
lowFramerate/port_0_17fps	2999	480	640	31.35	28.17	15.75	12.39
lowFramerate/tramCrossroad_1fps	899	350	640	39.51	36.73	21.37	16.71
nightVideos/busyBoulvard	2759	364	640	38.86	36.00	20.90	16.43
nightVideos/bridgeEntry	2499	430	630	34.52	31.85	17.90	14.20
nightVideos/winterStreet	1784	420	624	35.92	32.57	18.26	14.52
nightVideos/streetCornerAtNight	5199	245	595	52.86	51.56	32.83	25.35
PTZ/twoPositionPTZCam	2299	340	570	43.30	41.11	17.91	18.92
PTZ/intermittentPan	3499	368	560	40.65	38.52	16.39	17.71
badWeather/skating	3899	360	540	43.00	40.79	17.20	19.08
nightVideos/tramStation	2999	295	480	53.15	52.62	33.46	26.15
dynamicBackground/fountain01	1183	288	432	55.38	57.06	37.95	30.16
dynamicBackground/fountain02	1498	288	432	56.56	56.97	38.22	30.49
intermittentObjectMotion/abandonedBox	4499	288	432	55.64	58.03	38.19	30.26
intermittentObjectMotion/tramstop	3199	288	432	56.58	57.56	38.03	28.99
shadow/peopleInShade	1198	244	380	67.31	71.54	49.84	41.11
baseline/office	2049	240	360	71.72	75.86	37.85	45.39
baseline/pedestrians	1098	240	360	70.75	75.87	40.13	45.66
shadow/bungalows	1699	240	360	69.25	75.88	55.53	45.12
shadow/busStation	1249	240	360	69.51	74.76	55.41	46.24
cameraJitter/boulevard	2499	240	352	70.44	75.44	39.33	42.60
cameraJitter/sidewalk	1199	240	352	69.44	75.95	38.54	42.36
shadow/cubicle	7399	240	352	71.78	77.42	57.46	46.65
thermal/park	599	288	352	57.07	58.39	43.26	36.78
PTZ/zoomInZoomOut	1129	240	320	72.70	80.89	41.94	44.97
baseline/highway	1699	240	320	74.12	82.40	42.85	46.89
cameraJitter/traffic	1569	240	320	72.87	81.54	40.97	45.95
dynamicBackground/boats	7998	240	320	73.14	82.14	60.35	47.99
dynamicBackground/canoe	1188	240	320	72.64	81.62	61.19	46.91
dynamicBackground/overpass	2999	240	320	72.85	82.26	61.72	50.00
intermittentObjectMotion/parking	2499	240	320	73.56	81.88	61.94	51.23
intermittentObjectMotion/sofa	2749	240	320	73.57	82.95	62.15	47.82
intermittentObjectMotion/streetLight	3199	240	320	72.77	82.48	62.27	48.13
intermittentObjectMotion/winterDriveway	2499	240	320	74.19	82.81	62.96	48.48
lowFramerate/turnpike_0_5fps	1499	240	320	73.10	82.38	60.99	46.52
shadow/backdoor	1999	240	320	73.73	83.84	62.82	51.77
thermal/corridor	5399	240	320	74.86	83.65	62.48	51.11
thermal/diningRoom	3699	240	320	74.42	83.86	62.34	50.98
thermal/lakeSide	6499	240	320	74.93	83.60	63.23	51.68
thermal/library	4899	240	320	75.79	83.72	62.17	47.23

Best results are highlighted using grey.

the CDnet2012 and CDnet2014 datasets.

The average frame rate for processing images with the native resolution of 720×480 per algorithm was i) NeuroHSMDv2 28.71 fps, NeuroHSMDv1 24.95 fps, HSMDv2 12.37 fps and HSMDv1 11.25 fps. These results are in line with the results obtained for the 4 algorithms when tested against the CDnet2012 dataset.

5.3.3 Benchmark

Table 5.5 shows the results obtained after testing the 4 methods against the CDnet2012 ground-truth images using the scripts provided by Nil Goyette et al. [9].

Table 5.5 CDnet2012 Overall ranks

Method	$\overline{RC}$ ↓	Re ↑	Sp ↑	FPR ↓	FNR ↓	WCR ↓	CCR ↑	F1 ↑	Pr ↑
HSMDv1	1	0.52	0.994	0.006	0.23	0.024	0.976	0.77	0.62
HSMDv2	1	0.52	0.994	0.006	0.23	0.024	0.976	0.77	0.62
NeuroHSMDv1	1	0.52	0.994	0.006	0.23	0.024	0.976	0.77	0.62
NeuroHSMDv2	1	0.52	0.994	0.006	0.23	0.024	0.976	0.77	0.62

↑: the highest score is the best.
↓: the lowest result is the best.
All the 4 methods were ranked first because no changes were made to the customised SNN. *Re* stands for Recall, *Sp* Specificity, FPR False Positive Rate, FNR False Negative Rate, WCR Wrong Classifications Rate, CCR Correct Classifications Rate, Pr Precision, F1 F-score and $\overline{RC}$ Average Ranking across all Categories.

From the results shown in Table 5.5 it is possible to infer that the results obtained with the four methods are equivalent because all the methods were ranked in first place with the same values per

metric. Indexing all the algorithms in the first place was expected because the speed optimisation in version 2 of the NeuroHSMD and HSMD should not interfere with the model dynamics.

Table 5.6 depicts the results obtained after testing 4 methods against the CDnet2014 ground-truth images using the scripts provided by Nil Goyette et al. [9].

Table 5.6 CDnet2014 Overall ranks

Method	$\overline{RC}$ ↓	Re ↑	Sp ↑	FPR ↓	FNR ↓	WCR ↓	CCR ↑	F1 ↑	Pr ↑
HSMDv1	1	0.55	0.993	0.007	0.35	0.018	0.982	0.65	0.60
HSMDv2	1	0.55	0.993	0.007	0.35	0.018	0.982	0.65	0.60
NeuroHSMDv1	1	0.55	0.993	0.007	0.35	0.018	0.982	0.65	0.60
NeuroHSMDv2	1	0.55	0.993	0.007	0.35	0.018	0.982	0.65	0.60

↑: the highest score is the best.
↓: the lowest result is the best.
All the 4 methods were ranked first because no changes were made to the customised SNN. *Re* stands for Recall, *Sp* Specificity, FPR False Positive Rate, FNR False Negative Rate, WCR Wrong Classifications Rate, CCR Correct Classifications Rate, Pr Precision, F1 F-score and $\overline{RC}$ Average Ranking across all Categories.

From the results shown in Table 5.6 it is possible to infer that the results obtained with the four methods are probably the same because the four methods were ranked, again, in first place with the same values per metric. These results are important because it is possible to infer that there has been no degradation in accuracy as a consequence of the hardware acceleration.

5.4 Discussion

Two bio-inspired NeuroHSMD have been proposed to accelerate the HSMD algorithm that was discussed in Chapter4. The NeuroHSMDv1 and NeuroHSMDv2 (speed optimisation) were tested against the CDnet2012 and CDnet2014 datasets. The NeuroHSMDv1 has lower latency when processing images with resolutions equal to or greater than 480×295. The NeuronHSMDv2 (speed optimisation) has a lower latency when processing images with resolutions smaller than 480×295. To ensure a fair comparison between the software and hardware implementations, two HSMD versions were used (HSMDv1 is the same algorithm described in Chapter 4 and HSMDv2 with speed optimisation).

The HSMDv1, HSMDv2, NeuroHSMDv1 and NeuroHSMDv2 were all tested on the same computer equipped with a quad-core Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 16GB of DDR3 @ 1600 MHz and 1TB of HDD. The average frame rate for processing images with the native resolution of 720×480 per algorithm was:

CDnet2012:

i) NeuroHSMDv2 28.06 fps, NeuroHSMDv1 25.45 fps, HSMDv2

11.19 fps and HSMDv1 9.97 fps

CDnet2014:

i) NeuroHSMDv2 28.71 fps, NeuroHSMDv1 24.95 fps, HSMDv2 12.37 fps and HSMDv1 11.25 fps

The four methods were also tested against the ground-truth images available in the CDnet2012 and CDnet2014 datasets using the eight metrics, which were used to assess and compare the quality of the HSMD algorithm. The four methods obtained the same values for all the metrics and were all ranked first. The first place acquired by the four methods is an indication that there was no degradation in the hardware acceleration.

Future work includes optimising the HSMD algorithm to detect and track motion in challenging scenarios (e.g. low frame rate, dynamic background, and camera jitter) and an investigation to verify if the SNN improves the remaining methods' output. It is also planned to implement and evaluate more complex retinal cells (such as predictive cells) using SNNs, as well as assess the effectiveness of BS algorithms in optimising and accelerating such complex retinal

cells using FPGA technology.

Chapter 6

Discussion and Future work

6.1 Main contributions

Retinal cells are highly efficient in performing primary steps in processing of natural images. The retina performs visual tasks such as object movement sensitivity, looming, fast response to fast stimuli, and even prediction.

The first goal of the PhD research programme (see chapter 2) was to investigate and replicate the basic functionalities of DSGCs (such as detecting horizontal and vertical movements and more generic object motions). This PhD research programme's second objective was to explore the use of SNNs for implementing bio-inspired object motion detectors inspired by the OMS-GC. The third and final objective was to optimise the object motion detector, using FPGA technology to improve the latency by accelerating SNN and reducing the dynamic power consumption associated with high-frequency

clocks.

The MHSNN architecture presented in Chapter 3 contributes to the first and second objectives of this PhD research programme. MHSNN's main contributions are:

1. Layer 1 edge detection
2. Extraction of direction motion features in Layer 2;
3. Extraction of movement features in Layer 3;
4. Movement detection in Layer 4.

Therefore, the MHSNN outputs reflect object motion detection in vertebrate retinas. The most straightforward behaviour shown by DSGCs, horizontal and vertical movement detection, was modelled. Experiments were carried out using a semisynthetic dataset of a black cylinder performing leftwards, rightwards, downwards and upwards movement. A CODD algorithm that combined the seven BS available in the OpenCV library was implemented to benchmark against the MHSNN. The MHSNN was ranked first in terms of the PCC and exhibited the lowest PWC detecting the motion direction when tested against the semisynthetic dataset. Although the MHSNN could be adapted for modelling other types of gan-

glion cells, such as looming (sensitive to approach and recede movements), fast-response (sensitive to fast movements) and predictive (predicting movement trajectories) cells, the MHSNN is not a scalable architecture because of the required number of neurons and synapses (above 17000 neurons and 173700 synapses for processing 40×40 pixels image, see Chapter 3). Nevertheless, the number of neurons and synapses can be substantially decreased by combining SNN with existing BS algorithms.

The MHSNN architecture lead to the development of a bio-inspired HSMD discussed in Chapter 4, to detect object motion and assess against the CDnet2012 and CDnet2014 datasets. The HSMD contributes to the first and second objectives of this PhD research programme. These incorporate video sequences of many moving objects under various challenging environmental conditions and are widely used for benchmarking background subtraction algorithms. The CDnet2012 is composed of five categories of movements, and the CDnet2014 augments the initial five to eleven categories of movements. Eight metrics, utilised as standard in the change detection datasets, were used to assess and compare the quality of the HSMD algorithm. The HSMD algorithm performed overall best in both the CDnet2012 and CDnet2014 while perform-

ing better than all the tested BS algorithms in the intermittent object motion, night videos, thermal and turbulence categories, it performs second-best in the bad weather category and the third-best in the baseline and shadow categories. The comparatively good results are a consequence of using the SNN for emulating the basic functionality of OMS-GC, which improves the sensitivity of the HSMD to object motion. The HSMD is also the first hybrid SNN algorithm capable of processing video/image sequences in near real-time (i.e. 720×480@13.82fps [CDnet2014] and 720×480@13.92fps [CDnet2012]).

Finally, in Chapter 5, two bio-inspired NeuroHSMD have been proposed to accelerate the HSMD algorithm that was discussed in Chapter4. The NeuroHSMD algorithm contributes to all the objectives of this PhD research programme. The NeuroHSMDv1 and NeuroHSMDv2 (speed optimisation) were tested against the CDnet2012 and CDnet2014 datasets. The NeuroHSMDv2 (speed optimisation) has lower latency when processing higher resolution images (equal to or greater than 480×295). The NeuronHSMDv1 has a lower latency when processing images with lower resolutions (smaller or equal to 480×295). Both versions of the NeuroHSMD algorithms have produced the same results as the original HSMD

algorithm reported in Chapter 4. The average frame rate for processing images with the native resolution of 720×480 per algorithm was

CDnet2012:

i) NeuroHSMDv2 28.06 fps, NeuroHSMDv1 25.45 fps, HSMDv2 11.19 fps and HSMDv1 9.97 fps

CDnet2014:

i) NeuroHSMDv2 28.71 fps, NeuroHSMDv1 24.95 fps, HSMDv2 12.37 fps and HSMDv1 11.25 fps.

6.2 Future work

Future work on the MHSNN includes replacing the first two layers by BS algorithms to reduce the number of neurons and synapses and test it on natural images. It is also planned to adapt the MHSNN to detect approach and receding movements. At the same time, it is also planned to benchmark the MHSNN's SNN inference on CPUs, FPGAs and GPUs and compare the power consumption profiles of each solution. The power consumption and the PCC will be used to justify the technology (or technologies) to be utilised to design intelligent cameras to run variants of the MHSNN.

HSMD future work includes optimising the HSMD algorithm to

detect and track motion in challenging scenarios (e.g. low frame rate, dynamic background, and camera jitter). It is further planned to investigate if the SNN used in the HSMD also improves other OpenCV BS methods. Furthermore, the author will also test if the SNN can enhance other BS algorithms available on the OpenCV library. Simultaneously, it is also planned to benchmark the HSMD's SNN inference on CPUs, FPGAs and GPUs and compare the different power consumption profiles. The same methodology that was used to develop the HSMD algorithm will also be used to model more complex retinal cells such as the fast-response and predictive cells [18].

Future work for NeuroHSMD includes optimising the custom SNN to fit into low-cost Processor System (PS) and FPGA on the same chip (PS-FPGA), which could deliver the next-generation of intelligent cameras. The author also aims to convert the OpenCL kernels into Compute Unified Device Architecture (CUDA) kernels and evaluate the NeuroHSMD performance on a processor system and GPU on the same chip (PS-GPU). This will lead to development of new intelligent motion detectors which can be used in different fields from crowd monitoring to ADAS. The knowledge gathered in this PhD research programme about porting complex SNN into

FPGAs using OpenCL will also be used for porting complex retinal cells (e.g. fast-response and predictive GCs) and other parallelisable algorithms that can be accelerated using dedicated hardware.

Appendix

The MHSNN architecture presented in Chapter 3 was published on the Conference proceedings of the International Joint Conferences on Neural Networks 2018 [293].
P. Machado, A. Oikonomou, G. Cosma and T. M. McGinnity, "Bio-Inspired Ganglion Cell Models for Detecting Horizontal and Vertical Movements" 2018 International Joint Conference on Neural Networks (IJCNN), 2018, pp. 1-8, doi: 10.1109/IJCNN.2018.8489439.

The HSMD algorithm presented in Chapter 4 was published on the IEEE Access [318]. P. Machado, A. Oikonomou, J. F. Ferreira and T. M. McGinnity, "HSMD: An object motion detection algorithm using a Hybrid Spiking Neural Network Architecture," in IEEE Access, doi: 10.1109/ACCESS.2021.3111005.

The NeuroHSMD algorithm presented in Chapter 5 was submitted to the ACM Transactions on Reconfigurable Technology and Systems.

References

[1] H. Kolb, "Simple anatomy of the retina by helga kolb. webvision," 2011. Available online, https://webvision.med.utah.edu/book/part-i-foundations/simple-anatomy-of-the-retina/, last accessed: 2021-04-09. xi, 12, 13

[2] Chegg, "Learn About Structure Of Retina — Chegg.com." Available online, https://www.chegg.com/learn/biology/anatomy-physiology-in-biology/structure-of-retina, last accessed: 2021-06-26. xi, 14

[3] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952. xi, 16, 17

[4] W. Gerstner and W. M. Kistler, *Spiking Neuron Models*. Cambridge: Cambridge University Press, 2002. xii, 16, 17, 18, 19, 120

[5] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004. xii, 18, 19, 20

[6] P. Martinez-Canada, C. Morillas, B. Pino, E. Ros, and F. Pelayo, "A computational framework for realistic retina modeling," *International Journal of Neural Systems*, vol. 26, no. 07, p. 1650030, 2016. xii, 21, 22, 23

[7] H. Kolb, E. Fernandez, and R. Nelson, "The organization of the retina and visual system," *Webvision-The Organization of the Retina and Visual System*, 2005. xiii, 15, 78, 79, 88

[8] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting," *Neural computation*, vol. 22, no. 2, pp. 467–510, 2010. xiii, 66, 89, 90

[9] N. Goyette, P.-M. Jodoin, F. Porikli, J. Konrad, and P. Ishwar, "Changedetection. net: A new change detection benchmark dataset," in *2012 IEEE computer society conference on computer vision and pattern recognition workshops*, pp. 1–8, IEEE, 2012. xv, 8, 9, 28, 39, 46, 47, 118, 127, 129, 130, 169, 177, 178

[10] Intel, "Intel Stratix 10 Logic Array Blocks and Adaptive Logic Modules User Guide," 2020. Available online, https://www.intel.com/content/www/us/en/docs/programmable/683665/current/logic-array-blocks-and-adaptive-logic-24877.html, last accessed: 2021-04-02. xvii, 151

[11] Intel, "FPGA vs. GPU for Deep Learning Applications – Intel." xvii, 146, 153

[12] Intel, "Intel Stratix 10 Device Design Guidelines," 2020. Available online, https://www.intel.com/content/www/us/en/docs/programmable/683738/current/device-design-guidelines.html, last accessed: 2021-04-02. xvii, 153

[13] Intel, "Intel FPGA SDK for OpenCL Pro Edition: Programming Guide," 2021. Available online, https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807965224.html, last accessed: 2021-04-09. xvii, 157, 158, 159

[14] Terasic, "Terasic - DE10-Pro," 2021. Available onine, https://www.terasic.com.tw/cgi-bin/page/archive.

pl?Language=English&CategoryNo=13&No=1144&PartNo=2, last accessed: 2021-07-31. xvii, 159, 160

[15] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 2, pp. 1–35, 2019. xix, 64, 65

[16] M.-N. Chapel and T. Bouwmans, "Moving objects detection with a moving camera: A comprehensive review," *Computer science review*, vol. 38, p. 100310, 2020. 1, 2, 3, 29, 32, 36, 37, 44, 49, 115, 116, 117

[17] B. Garcia-Garcia, T. Bouwmans, and A. J. R. Silva, "Background subtraction in real applications: Challenges, current models and future directions," *Computer Science Review*, vol. 35, p. 100204, 2020. 1, 29, 32, 36, 37, 115, 116

[18] T. Gollisch and M. Meister, "Eye smarter than scientists believed: neural computations in circuits of the retina," *Neuron*, vol. 65, no. 2, pp. 150–164, 2010. 2, 15, 29, 78, 88, 118, 119, 187

[19] C. Trujillo Herrera and J. G. Labram, "A perovskite

retinomorphic sensor," *Applied Physics Letters*, vol. 117, p. 233501, dec 2020. 2

[20] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 2, pp. 1–35, 2019. 2, 3

[21] H. Kolb, "How the retina works: Much of the construction of an image takes place in the retina itself through the use of specialized neural circuits," *American scientist*, vol. 91, no. 1, pp. 28–35, 2003. 2, 29, 80

[22] B. Garcia-Garcia, T. Bouwmans, and A. J. R. Silva, "Background subtraction in real applications: Challenges, current models and future directions," *Computer Science Review*, vol. 35, p. 100204, 2020. 2, 3, 29, 36

[23] U. Illahi and M. S. Mir, "Comparative analysis of background subtraction and cnn algorithms for mid-block traffic data collection and classification," *Int J Math Eng Manag Sci*, vol. 5, no. 6, pp. 1440–1451, 2020. 2, 3

[24] G. Rebala, A. Ravi, and S. Churiwala, "Machine learning def-

inition and basics," in *An Introduction to Machine Learning*, pp. 1–17, Springer, 2019. 3

[25] J. Von Neumann, "First draft of a report on the edvac," *Moore School, University of Pennsylvania*, 1945. 4

[26] M. ZHANG, G. Zonghua, and P. Gang, "A survey of neuromorphic computing based on spiking neural networks," *Chinese Journal of Electronics*, vol. 27, no. 4, pp. 667–674, 2018. 4

[27] S. Blank, "What the GlobalFoundries' Retreat Really Means - IEEE Spectrum," 2018. Available online, https://spectrum.ieee.org/nanoclast/semiconductors/devices/what-globalfoundries-retreat-really-means, last accessed: 2021-07-21. 4

[28] Intel, "Intel FPGA SDK for OpenCL Pro Edition Best Practices Guide," 2021. Available online, https://www.intel.com/content/www/us/en/docs/programmable/683521/21-4/introduction-to-pro-edition-best-practices.html, last accessed: 2021-04-02. 4, 63, 154

[29] Xilinx and Inc, " AXI Reference Guide UG761," 2011. Available online, https://www.xilinx.com/support/documents/

ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf, last accessed: 2021-07-21. 4

[30] V. Samsonov, "Improvement of the background subtraction algorithm," 2017. Available online, https://summerofcode.withgoogle.com/archive/2017/projects/6453014550282240/, last accessed: 23/11/2020. 8, 39, 117, 118, 120

[31] OpenCV, "OpenCV: Operations on arrays," 2021. Available online, https://docs.opencv.org/3.4/d4/dd5/classcv_1_1bgsegm_1_1BackgroundSubtractorGSOC.html, last accessed: 2021-05-27. 8, 9, 39, 117

[32] Y. Wang, P.-M. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar, "Cdnet 2014: An expanded change detection benchmark dataset," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 387–394, 2014. 8, 28, 38, 39, 47, 118, 127, 129, 130, 169

[33] Intel, "Using Intel FPGA SDK for OpenCL on DE-Series Boards," 2019. Available online, https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/OpenCL_On_DE_Series_Boards.pdf, last accessed: 2021-04-02. 9, 149, 154

[34] Kronos, "OpenCL Overview," 2021. Available online, https://www.khronos.org/opencl/, last accessed: 2021-04-02. 9, 149

[35] R. Cajal, "The Structure of the Retina," *hTorpe SA, Glickstein M, translators.*, 1892. 12

[36] M. Piccolino, "Cajal and the retina: a 100-year retrospective," *Trends in neurosciences*, vol. 11, no. 12, pp. 521–525, 1988. 12

[37] R. H. Masland, "The fundamental plan of the retina," *Nature neuroscience*, vol. 4, no. 9, pp. 877–886, 2001. 12

[38] B. Bosze, R. B. Hufnagel, and N. L. Brown, "Chapter 21 - specification of retinal cell types," in *Patterning and Cell Type Specification in the Developing CNS and PNS (Second Edition)* (J. Rubenstein, P. Rakic, B. Chen, and K. Y. Kwan, eds.), pp. 481–504, Academic Press, second edition ed., 2020. 13

[39] M. M. Garvert and T. Gollisch, "Local and global contrast adaptation in retinal ganglion cells," *Neuron*, vol. 77, no. 5, pp. 915–928, 2013. 15

[40] T. Guo, D. Tsai, S. Bai, J. W. Morley, G. J. Suaning, N. H. Lovell, and S. Dokos, "Understanding the retina: A review of

computational models of the retina from the single cell to the network level," *Critical Reviews™ in Biomedical Engineering*, vol. 42, no. 5, 2014. 15, 16, 33

[41] P. A. Roberts, E. A. Gaffney, P. J. Luthert, A. J. Foss, and H. M. Byrne, "Mathematical and computational models of the retina in health, development and disease," *Progress in retinal and eye research*, vol. 53, pp. 48–69, 2016. 15

[42] P. J. Vance, G. P. Das, D. Kerr, S. A. Coleman, T. M. McGinnity, T. Gollisch, and J. K. Liu, "Bioinspired approach to modeling retinal ganglion cells using system identification techniques," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 5, pp. 1796–1808, 2017. 15

[43] Q. Fu, H. Wang, C. Hu, and S. Yue, "Towards computational models and applications of insect visual systems for motion perception: A review," *Artificial life*, vol. 25, no. 3, pp. 263–311, 2019. 15

[44] N. K. Kühn and T. Gollisch, "Activity correlations between direction-selective retinal ganglion cells synergistically enhance motion decoding from complex visual scenes," *Neuron*, vol. 101, no. 5, pp. 963–976, 2019. 15

[45] J. Nagumo, S. Arimoto, and S. Yoshizawa, "Impulses and physiological states in models of nerve membrane," in *Proc. Inst. Radio Engrs*, vol. 50, pp. 2061–2070, 1962. 17

[46] C. Morris and H. Lecar, "Voltage oscillations in the barnacle giant muscle fiber," *Biophysical journal*, vol. 35, no. 1, pp. 193–213, 1981. 17

[47] R. Rose and J. Hindmarsh, "The assembly of ionic currents in a thalamic neuron i. the three-dimensional model," *Proceedings of the Royal Society of London. B. Biological Sciences*, vol. 237, no. 1288, pp. 267–288, 1989. 17

[48] A. O. Komendantov and N. I. Kononenko, "Deterministic chaos in mathematical model of pacemaker activity in bursting neurons of snail, helix pomatia," *Journal of theoretical biology*, vol. 183, no. 2, pp. 219–230, 1996. 17, 72

[49] H. R. Wilson, "Simplified dynamics of human and mammalian neocortical neurons," *Journal of theoretical biology*, vol. 200, no. 4, pp. 375–388, 1999. 17

[50] F. Pelayo, A. Martinez, S. Romero, C. A. Morillas, E. Ros, and E. Fernandez, "Cortical visual neuro-prosthesis for the blind: retina-like software/hardware preprocessor," in *First*

International IEEE EMBS Conference on Neural Engineering, 2003. Conference Proceedings., pp. 150–153, IEEE, 2003. 20

[51] F. J. Pelayo, S. Romero, C. A. Morillas, A. Martinez, E. Ros, and E. Fernandez, "Translating image sequences into spike patterns for cortical neuro-stimulation," *Neurocomputing*, vol. 58, pp. 885–892, 2004. 20

[52] E. CORDIS, "Cortical visual neuroprosthesis for the blind — CORTIVIS Project — FP5 — CORDIS — European Commission." Available online, https://cordis.europa.eu/project/id/QLK6-CT-2001-00279, last accessed: 2021-06-24. 20

[53] J. Jordan, H. Mørk, S. B. Vennemo, D. Terhorst, A. Peyser, T. Ippen, R. Deepu, J. M. Eppler, A. van Meegen, S. Kunkel, A. Sinha, T. Fardet, S. Diaz, A. Morrison, W. Schenck, D. Dahmen, J. Pronold, J. Stapmanns, G. Trensch, S. Spreizer, J. Mitchell, S. Graber, J. Senk, C. Linssen, J. Hahne, A. Serenko, D. Naoumenko, E. Thomson, I. Kitayama, S. Berns, and H. E. Plesser, "Nest 2.18.0," June 2019. Available online, https://doi.org/10.5281/zenodo.2605422, last accessed: 11/05/2022. 22, 23, 124

[54] R. A. Tikidji-Hamburyan, V. Narayana, Z. Bozkus, and T. A.

El-Ghazawi, "Software for brain network simulations: a comparative study," *Frontiers in Neuroinformatics*, vol. 11, p. 46, 2017. 22, 24

[55] D. F. Goodman and R. Brette, "The brian simulator," *Frontiers in neuroscience*, vol. 3, p. 26, 2009. 22, 23, 26, 67

[56] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *Elife*, vol. 8, p. e47314, 2019. 23

[57] J. L. Lobo, J. Del Ser, A. Bifet, and N. Kasabov, "Spiking neural networks and online learning: An overview and perspectives," *Neural Networks*, vol. 121, pp. 88–100, 2020. 24

[58] D. Rasmussen, "Nengodl: Combining deep learning and neuromorphic modelling methods," *Neuroinformatics*, vol. 17, no. 4, pp. 611–628, 2019. 24

[59] L. Long and A. Gupta, "Biologically-inspired spiking neural networks with hebbian learning for vision processing," in *46th AIAA Aerospace Sciences Meeting and Exhibit*, p. 885, 2008. 24

[60] Q. Wu, T. M. McGinnity, L. P. Maguire, A. Belatreche, and B. Glackin, "Processing visual stimuli using hierarchical spik-

ing neural networks," *Neurocomputing*, vol. 71, no. 10-12, pp. 2055–2068, 2008. 24, 25

[61] Q. Wu, T. M. McGinnity, L. Maguire, J. Cai, and G. D. Valderrama-Gonzalez, "Motion detection using spiking neural network model," in *International conference on intelligent computing*, pp. 76–83, Springer, 2008. 24, 25, 28

[62] R. Cai, Q. Wu, P. Wang, H. Sun, and Z. Wang, "Moving target detection and classification using spiking neural networks," in *International Conference on Intelligent Science and Intelligent Data Engineering*, pp. 210–217, Springer, 2011. 25

[63] J. Zylberberg, J. T. Murphy, and M. R. DeWeese, "A sparse coding model with synaptically local plasticity and spiking neurons can account for the diverse shapes of v1 simple cell receptive fields," *PLoS computational biology*, vol. 7, no. 10, p. e1002250, 2011. 25

[64] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, no. 3, pp. 267–273, 1982. 25

[65] D. Kerr, S. A. Coleman, T. M. McGinnity, and M. Clogenson, "Biologically inspired intensity and range image feature

extraction," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2013. 25

[66] D. Kerr, T. M. McGinnity, S. Coleman, and M. Clogenson, "A biologically inspired spiking model of visual processing for image feature detection," *Neurocomputing*, vol. 158, pp. 268–280, 2015. 26

[67] A. Tavanaei and A. S. Maida, "Bio-inspired spiking convolutional neural network using layer-wise sparse coding and stdp learning," *arXiv preprint arXiv:1611.03000*, 2016. 26

[68] E. Rueckert, D. Kappel, D. Tanneberg, D. Pecevski, and J. Peters, "Recurrent spiking networks solve planning tasks," *Scientific reports*, vol. 6, no. 1, pp. 1–10, 2016. 26

[69] Q. Y. Sun, Q. X. Wu, X. Wang, and L. Hou, "A spiking neural network for extraction of features in colour opponent visual pathways and fpga implementation," *Neurocomputing*, vol. 228, pp. 119–132, 2017. 27, 28

[70] P. Machado, G. Cosma, and T. M. McGinnity, "Natcsnn: A convolutional spiking neural network for recognition of objects extracted from natural images," *Lecture Notes in Computer Science*, p. 351–362, 2019. 27, 28

[71] P. K. Tadiparthi, S. Ponnada, K. Jhansi, P. K. Bheemavarapu, and A. Gottimukkala, "A comprehensive review of moving object identification using background subtraction in dynamic scenes," *Solid State Technology*, vol. 64, no. 2, pp. 4114–4124, 2021. 29

[72] T. Bouwmans, F. El Baf, and B. Vachon, "Statistical background modeling for foreground detection: A survey," in *Handbook of pattern recognition and computer vision*, pp. 181–199, World Scientific, 2010. 29, 36

[73] T. Bouwmans, "Traditional and recent approaches in background modeling for foreground detection: An overview," *Computer science review*, vol. 11, pp. 31–66, 2014. 29, 36

[74] T. Bouwmans, F. Porikli, B. Höferlin, and A. Vacavant, *Background modeling and foreground detection for video surveillance*. CRC press, 2014. 29, 36

[75] G. Monteiro, J. Marcos, M. Ribeiro, and J. Batista, "Robust segmentation for outdoor traffic surveillance," in *2008 15th IEEE International Conference on Image Processing*, pp. 2652–2655, IEEE, 2008. 32

[76] G. Monteiro, J. Marcos, M. Ribeiro, and J. Batista, "Robust segmentation process to detect incidents on highways,"

in *International Conference Image Analysis and Recognition*, pp. 110–121, Springer, 2008. 32

[77] G. Monteiro, J. Marcos, and J. Batista, "Stopped vehicle detection system for outdoor traffic surveillance," *RECPAD 2008*, 2008. 32

[78] G. Monteiro, *Traffic video surveillance for automatic incident detection on highways*. PhD thesis, Coimbra, Portugal: University of Coimbra, 2008. 32

[79] J. Batista, P. Peixoto, C. Fernandes, and M. Ribeiro, "A dual-stage robust vehicle detection and tracking for real-time traffic monitoring," in *2006 IEEE Intelligent Transportation Systems Conference*, pp. 528–535, IEEE, 2006. 32

[80] R. A. Hadi, L. E. George, and M. J. Mohammed, "A computationally economic novel approach for real-time moving multi-vehicle detection and tracking toward efficient traffic surveillance," *Arabian Journal for Science and Engineering*, vol. 42, no. 2, pp. 817–831, 2017. 32

[81] Á. Virginás-Tar, M. Baba, V. Gui, D. Pescaru, and I. Jian, "Vehicle counting and classification for traffic surveillance using wireless video sensor networks," in *2014 22nd Telecom-*

munications Forum Telfor (TELFOR), pp. 1019–1022, IEEE, 2014. 32

[82] R. Lin, X. Cao, Y. Xu, C. Wu, and H. Qiao, "Airborne moving vehicle detection for video surveillance of urban traffic," in *2009 IEEE Intelligent Vehicles Symposium*, pp. 203–208, IEEE, 2009. 32

[83] O. Masoud and N. P. Papanikolopoulos, "A novel method for tracking and counting pedestrians in real-time using a single camera," *IEEE transactions on vehicular technology*, vol. 50, no. 5, pp. 1267–1278, 2001. 33

[84] L. Yi, Y. Ting, and L. Xinqiao, "Analysis and simulation of crowd in airport multiple transport modes," in *Proceedings of the 2020 International Conference on Aviation Safety and Information Technology*, pp. 36–41, 2020. 33

[85] F. Z. Qureshi, "Object-video streams for preserving privacy in video surveillance," in *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, pp. 442–447, IEEE, 2009. 33

[86] D.-J. Wang, W.-S. Chen, T.-H. Chen, and T.-Y. Chen, "The study on ship-flow analysis and counting system in a specific

sea-area based on video processing," in *2008 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 655–658, IEEE, 2008. 33

[87] S. Chen, Z. Feng, Q. Lu, B. Mahasseni, T. Fiez, A. Fern, and S. Todorovic, "Play type recognition in real-world football video," in *IEEE Winter Conference on Applications of Computer Vision*, pp. 652–659, IEEE, 2014. 33

[88] L. H. Leong, M. A. Zulkifley, and A. B. Hussain, "Computer vision approach to automatic linesman," in *2014 IEEE 10th International Colloquium on Signal Processing and its Applications*, pp. 212–215, IEEE, 2014. 33

[89] S. Guler and M. K. Farrow, "Abandoned object detection in crowded places," in *Proc. of PETS*, pp. 18–23, Citeseer, 2006. 33

[90] B. Li, J. Zhang, Z. Zhang, and Y. Xu, "A people counting method based on head detection and tracking," in *2014 International Conference on Smart Computing*, pp. 136–141, IEEE, 2014. 33

[91] E. Cermeño, A. Pérez, and J. A. Sigüenza, "Intelligent video surveillance beyond robust background modeling," *Expert Systems with Applications*, vol. 91, pp. 138–149, 2018. 33

[92] M. Dotter and J. Harguess, "Automated situational reporting," in *Geospatial Informatics XI*, vol. 11733, p. 117330G, International Society for Optics and Photonics, 2021. 33

[93] K. Ren, G. Bernes, M. Hetta, and J. Karlsson, "Tracking and analysing social interactions in dairy cattle with real-time locating system and machine learning," *Journal of Systems Architecture*, vol. 116, p. 102139, 2021. 33

[94] K. Pasupa, N. Pantuwong, and S. Nopparit, "A comparative study of feature point matching versus foreground detection for computer detection of dairy cows in video frames," *Artificial Life and Robotics*, vol. 20, no. 4, pp. 320–326, 2015. 33

[95] F. Okura, S. Ikuma, Y. Makihara, D. Muramatsu, K. Nakada, and Y. Yagi, "Rgb-d video-based individual identification of dairy cows using gait and texture analyses," *Computers and Electronics in Agriculture*, vol. 165, p. 104944, 2019. 33

[96] T. Fasciano, A. Dornhaus, and M. C. Shin, "Ant tracking with occlusion tunnels," in *IEEE Winter Conference on Applications of Computer Vision*, pp. 947–952, IEEE, 2014. 33

[97] T. Kimura, M. Ohashi, K. Crailsheim, T. Schmickl, R. Okada, G. Radspieler, and H. Ikeno, "Development of a new method

to track multiple honey bees with complex behaviors on a flat laboratory arena," *PloS one*, vol. 9, no. 1, p. e84656, 2014. 33

[98] C. Yang and J. Collins, "Improvement of honey bee tracking on 2d video with hough transform and kalman filter," *Journal of Signal Processing Systems*, vol. 90, no. 12, pp. 1639–1650, 2018. 33

[99] M. Wu, X. Cao, and S. Guo, "Accurate detection and tracking of ants in indoor and outdoor environments," *bioRxiv*, 2020. 33

[100] J. Zhao, Z. Gu, M. Shi, H. Lu, J. Li, M. Shen, Z. Ye, and S. Zhu, "Spatial behavioral characteristics and statistics-based kinetic energy modeling in special behaviors detection of a shoal of fish in a recirculating aquaculture system," *Computers and Electronics in Agriculture*, vol. 127, pp. 271–280, 2016. 33

[101] G. de Oliveira Feijó, V. A. Sangalli, I. N. L. da Silva, and M. S. Pinho, "An algorithm to track laboratory zebrafish shoals," *Computers in biology and medicine*, vol. 96, pp. 79–90, 2018. 33

[102] M. Srividya, R. Hemavathy, and G. Shobha, "Underwater video processing for detecting and tracking moving object,"

International Journal of Engineering and computer science, vol. 3, no. 5, pp. 5843–5847, 2014. 33

[103] H. Qin, Y. Peng, and X. Li, "Foreground extraction of underwater videos via sparse and low-rank matrix decomposition," in *2014 ICPR Workshop on Computer Vision for Analysis of Underwater Imagery*, pp. 65–72, IEEE, 2014. 33

[104] E. Hossain, S. S. Alam, A. A. Ali, and M. A. Amin, "Fish activity tracking and species identification in underwater video," in *2016 5th International conference on informatics, electronics and vision (ICIEV)*, pp. 62–66, IEEE, 2016. 33

[105] N. Seese, A. Myers, K. Smith, and A. O. Smith, "Adaptive foreground extraction for deep fish classification," in *2016 ICPR 2nd Workshop on Computer Vision for Analysis of Underwater Imagery (CVAUI)*, pp. 19–24, IEEE, 2016. 33

[106] A. Salman, S. Maqbool, A. H. Khan, A. Jalal, and F. Shafait, "Real-time fish detection in complex backgrounds using probabilistic background modelling," *Ecological Informatics*, vol. 51, pp. 44–51, 2019. 34

[107] N. Wawrzyniak, T. Hyla, and A. Popik, "Vessel detection and tracking method based on video surveillance," *Sensors*, vol. 19, no. 23, p. 5230, 2019. 34

[108] M. Fattahzadeh and A. Saghaei, "A statistical method for sequential images–based process monitoring," *International Journal of Engineering*, vol. 33, no. 7, pp. 1285–1292, 2020. 34

[109] H. Ghaffarian, P. Lemaire, Z. Zhi, L. Tougne, B. MacVicar, and H. Piégay, "Automated quantification of floating wood pieces in rivers from video monitoring: a new software tool and validation," *Earth Surface Dynamics*, vol. 9, no. 3, pp. 519–537, 2021. 34

[110] Y. Cao, Q. Tang, X. Wu, and X. Lu, "Effnet: Enhanced feature foreground network for video smoke source prediction and detection," *IEEE Transactions on Circuits and Systems for Video Technology*, 2021. 34

[111] V. Reno, N. Mosca, M. Nitti, T. D'Orazio, D. Campagnoli, A. Prati, and E. Stella, "Tennis player segmentation for semantic behavior analysis," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 1–8, 2015. 34

[112] Y. Liu and M. Zhi, "Moving object detection in tennis video," in *2015 8th International Conference on Intelligent Networks*

and Intelligent Systems (ICINIS), pp. 109–112, IEEE, 2015. 34

[113] W. Barhoumi, "Detection of highly articulated moving objects by using co-segmentation with application to athletic video sequences," *Signal, Image and Video Processing*, vol. 9, no. 7, pp. 1705–1715, 2015. 34

[114] W. Kim, S.-W. Moon, J. Lee, D.-W. Nam, and C. Jung, "Multiple player tracking in soccer videos: an adaptive multi-scale sampling approach," *Multimedia Systems*, vol. 24, no. 6, pp. 611–623, 2018. 34

[115] R. Meghana, Y. Chitkara, S. Apoorva, *et al.*, "Background-modelling techniques for foreground detection and tracking using gaussian mixture model," in *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 1129–1134, IEEE, 2019. 34

[116] W. Kim, "Multiple object tracking in soccer videos using topographic surface analysis," *Journal of Visual Communication and Image Representation*, vol. 65, p. 102683, 2019. 34

[117] B. K. Chakraborty, D. Sarma, M. K. Bhuyan, and K. F. MacDorman, "Review of constraints on vision-based gesture recog-

nition for human–computer interaction," *IET Computer Vision*, vol. 12, no. 1, pp. 3–15, 2018. 34

[118] P. K. Podder, M. Paul, and M. Murshed, "Foreground motion and spatial saliency-based efficient hevc video coding," in *2015 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pp. 1–6, IEEE, 2015. 34

[119] L. Zhao, X. Zhang, X. Zhang, S. Wang, S. Wang, S. Ma, and W. Gao, "Intelligent analysis oriented surveillance video coding," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 37–42, IEEE, 2017. 34

[120] H. Huang, X. Fang, Y. Ye, S. Zhang, and P. L. Rosin, "Practical automatic background substitution for live video," *Computational Visual Media*, vol. 3, no. 3, p. 1, 2017. 35

[121] M. Grega, P. Donath, P. Guzik, J. Król, A. Matiolański, K. Rusek, and A. Dziech, "Application of logistic regression for background substitution," in *International Conference on Multimedia Communications, Services and Security*, pp. 33–46, Springer, 2017. 35

[122] C. R. Del-Blanco, T. Mantecón, M. Camplani, F. Jaureguizar, L. Salgado, and N. García, "Foreground segmentation in depth

imagery using depth and spatial dynamic models for video surveillance applications," *Sensors*, vol. 14, no. 2, pp. 1961–1987, 2014. 35

[123] R. S. Dixit, S. Gandhe, and P. Dhulekar, "Pedestrian protection system for adas using arm 9," *International Journal of Computer Applications*, vol. 975, p. 8887, 2015. 35

[124] P. Vasuki and S. Veluchamy, "Pedestrian detection for driver assistance systems," in *2016 International Conference on Recent Trends in Information Technology (ICRTIT)*, pp. 1–4, IEEE, 2016. 35

[125] A. M. McIvor, "Background subtraction techniques," *Proc. of Image and Vision Computing*, vol. 4, pp. 3099–3104, 2000. 35

[126] M. Piccardi, "Background subtraction techniques: a review," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, vol. 4, pp. 3099–3104, IEEE, 2004. 35, 115

[127] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 19, no. 7, pp. 780–785, 1997. 36

[128] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell, "Towards robust automatic traffic scene analysis in real-time," in *Proceedings of 12th International Conference on Pattern Recognition*, vol. 1, pp. 126–131, IEEE, 1994. 36

[129] B. P. L. Lo and S. Velastin, "Automatic congestion detection system for underground platforms," in *Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing. ISIMP 2001 (IEEE Cat. No. 01EX489)*, pp. 158–161, IEEE, 2001. 36

[130] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati, "Detecting moving objects, ghosts, and shadows in video streams," *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, no. 10, pp. 1337–1342, 2003. 36

[131] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proceedings. 1999 IEEE computer society conference on computer vision and pattern recognition (Cat. No PR00149)*, vol. 2, pp. 246–252, IEEE, 1999. 36, 37, 38, 48, 127

[132] B. Han, D. Comaniciu, and L. Davis, "Sequential kernel density approximation through mode propagation: Applications

to background modeling," in *proc. ACCV*, vol. 4, pp. 818–823, Citeseer, 2004. 36

[133] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," in *European conference on computer vision*, pp. 751–767, Springer, 2000. 36

[134] M. Seki, T. Wada, H. Fujiwara, and K. Sumi, "Background subtraction based on cooccurrence of image variations," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2, pp. II–II, IEEE, 2003. 36

[135] N. M. Oliver, B. Rosario, and A. P. Pentland, "A bayesian computer vision system for modeling human interactions," *IEEE transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 831–843, 2000. 36, 44

[136] S.-C. S. Cheung and C. Kamath, "Robust background subtraction with foreground validation for urban traffic video," *EURASIP Journal on Advances in Signal Processing*, vol. 2005, no. 14, pp. 1–11, 2005. 35

[137] S. Y. Elhabian, K. M. El-Sayed, and S. H. Ahmed, "Moving object detection in spatial domain using background removal

techniques-state-of-art," *Recent patents on computer science*, vol. 1, no. 1, pp. 32–54, 2008. 35

[138] M. Cristani, M. Farenzena, D. Bloisi, and V. Murino, "Background subtraction for automated multisensor surveillance: a comprehensive review," *EURASIP Journal on Advances in signal Processing*, vol. 2010, pp. 1–24, 2010. 36

[139] A. Elgammal, "Background subtraction: Theory and practice," *Synthesis Lectures on Computer Vision*, vol. 5, no. 1, pp. 1–83, 2014. 36

[140] P. KaewTraKulPong and R. Bowden, "An improved adaptive background mixture model for real-time tracking with shadow detection," in *Video-based surveillance systems*, pp. 135–144, Springer, 2002. 37, 48

[141] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 2, pp. 28–31, IEEE, 2004. 38, 127

[142] Z. Zivkovic and F. Van Der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006. 38, 127

[143] S. Zeevi, "BackgroundSubtractorCNT: A Fast Background Subtraction Algorithm," Dec. 2016. Available online, https://doi.org/10.5281/zenodo.4267853, last accessed: 23/11/2020. 38, 127

[144] S. Zeevi, "Fastest background subtraction is BackgroundSubtractorCNT," 2016. Available onine, https://www.theimpossiblecode.com/blog/fastest-background-subtraction-opencv/, last accessed: 09/11/2020. 38

[145] A. B. Godbehere and K. Goldberg, "Algorithms for visual tracking of visitors under variable-lighting conditions for a responsive audio art installation," in *Controls and art*, pp. 181–204, Springer, 2014. 38, 127

[146] L. Guo, D. Xu, and Z. Qiang, "Background subtraction using local svd binary pattern," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 86–94, 2016. 39, 127

[147] OpenCV, "Background Subtractor GSOC," 2020. Available online, https://docs.opencv.org/4.5.0/d4/dd5/classcv_1_1bgsegm_1_1BackgroundSubtractorGSOC.html, last accessed: 06/11/2020. 39, 120

[148] V. Samsonov, "Improved background subtraction algorithm," Nov. 2017. Available online, https://zenodo.org/record/4269865, last accessed: 23/11/2020. 39, 118, 120, 127

[149] Google, "Google Summer of Code Archive," 2017. Available online, https://summerofcode.withgoogle.com/archive/2017/projects/, last accessed: 2021-07-28. 39

[150] M. Braham, S. Pierard, and M. Van Droogenbroeck, "Semantic background subtraction," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 4552–4556, Ieee, 2017. 40, 41

[151] D. Zeng, M. Zhu, and A. Kuijper, "Combining background subtraction algorithms with convolutional neural network," *Journal of Electronic Imaging*, vol. 28, no. 1, p. 013011, 2019. 40, 47

[152] X. Liang, S. Liao, X. Wang, W. Liu, Y. Chen, and S. Z. Li, "Deep background subtraction with guided learning," in *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, IEEE, 2018. 40

[153] D. Zeng and M. Zhu, "Background subtraction using multiscale fully convolutional network," *IEEE Access*, vol. 6, pp. 16010–16021, 2018. 40

[154] D. Zeng, X. Chen, M. Zhu, M. Goesele, and A. Kuijper, "Background subtraction with real-time semantic segmentation," *IEEE Access*, vol. 7, pp. 153869–153884, 2019. 41

[155] D. H. Parks and S. S. Fels, "Evaluation of background subtraction algorithms with post-processing," in *2008 IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance*, pp. 192–199, IEEE, 2008. 41

[156] S. N. Yaakob, Z. Kadim, H. H. Woon, *et al.*, "Moving object extraction in ptz camera using the integration of background subtraction and local histogram processing," in *2012 International Symposium on Computer Applications and Industrial Electronics (ISCAIE)*, pp. 167–172, IEEE, 2012. 42

[157] R. H. Chan, C.-W. Ho, and M. Nikolova, "Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization," *IEEE Transactions on image processing*, vol. 14, no. 10, pp. 1479–1485, 2005. 42

[158] R. T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, *et al.*, "A system for video surveillance and monitoring," *VSAM final report*, vol. 2000, no. 1-68, p. 1, 2000. 43

[159] C. Wren, "Real-time tracking of the human body," *Photonics East, SPIE*, vol. 2615, 1995. 43

[160] M. I. Chacon-Murguia and S. Gonzalez-Duarte, "An adaptive neural-fuzzy approach for object detection in dynamic backgrounds for surveillance systems," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 8, pp. 3286–3298, 2011. 43

[161] W. Hu, T. Tan, L. Wang, and S. Maybank, "A survey on visual surveillance of object motion and behaviors," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 34, no. 3, pp. 334–352, 2004. 43

[162] F. Diana and T. Bouwmans, "Background modeling via a supervised subspace learning," in *International Conference on Image, Video Processing and Computer Vision*, pp. 1–7, International Society for Research in Science and Technology, 2010. 44

[163] D. Farcas, C. Marghes, and T. Bouwmans, "Background subtraction via incremental maximum margin criterion: a discriminative subspace approach," *Machine Vision and Applications*, vol. 23, no. 6, pp. 1083–1101, 2012. 44

[164] C. Marghes and T. Bouwman, "Background modeling via in-

cremental maximum margin criterion," in *Asian Conference on Computer Vision*, pp. 394–403, Springer, 2010. 44

[165] E. J. Candes, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?," *Journal of the ACM (JACM)*, vol. 58, no. 3, pp. 1–37, 2011. 44

[166] A. Sobral, T. Bouwmans, and E.-h. ZahZah, "Double-constrained rpca based on saliency maps for foreground detection in automated maritime surveillance," in *2015 12th IEEE international conference on advanced video and signal based surveillance (AVSS)*, pp. 1–6, IEEE, 2015. 44

[167] S. Javed, S. K. Jung, A. Mahmood, and T. Bouwmans, "Motion-aware graph regularized rpca for background modeling of complex scenes," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 120–125, IEEE, 2016. 44

[168] S. Javed, A. Mahmood, T. Bouwmans, and S. K. Jung, "Spatiotemporal low-rank modeling for complex scene background initialization," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 6, pp. 1315–1329, 2016. 44

[169] J. He, L. Balzano, and A. Szlam, "Incremental gradient on the grassmannian for online foreground and background sepa-

ration in subsampled video," in *2012 ieee conference on computer vision and pattern recognition*, pp. 1568–1575, IEEE, 2012. 45

[170] H. Guo, C. Qiu, and N. Vaswani, "Practical reprocs for separating sparse and low-dimensional signal sequences from their sum—part 1," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4161–4165, IEEE, 2014. 45

[171] P. Rodriguez and B. Wohlberg, "Incremental principal component pursuit for video background modeling," *Journal of Mathematical Imaging and Vision*, vol. 55, no. 1, pp. 1–18, 2016. 45

[172] P. Narayanamurthy and N. Vaswani, "A fast and memory-efficient algorithm for robust pca (merop)," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4684–4688, IEEE, 2018. 45

[173] N. Vaswani, T. Bouwmans, S. Javed, and P. Narayanamurthy, "Robust subspace learning: Robust pca, robust subspace tracking, and robust subspace recovery," *IEEE signal processing magazine*, vol. 35, no. 4, pp. 32–55, 2018. 45

[174] S. Javed, P. Narayanamurthy, T. Bouwmans, and N. Vaswani, "Robust pca and robust subspace tracking: A comparative evaluation," in *2018 IEEE Statistical Signal Processing Workshop (SSP)*, pp. 836–840, IEEE, 2018. 45

[175] S. Prativadibhayankaram, H. V. Luong, T. H. Le, and A. Kaup, "Compressive online video background–foreground separation using multiple prior information and optical flow," *Journal of Imaging*, vol. 4, no. 7, p. 90, 2018. 45

[176] S. Javed, T. Bouwmans, and S. K. Jung, "Stochastic decomposition into low rank and sparse tensor for robust background subtraction," in *6th International Conference on Imaging for Crime Prevention and Detection (ICDP-15)*, pp. 1–6, IET, 2015. 45

[177] A. Sobral, S. Javed, S. Ki Jung, T. Bouwmans, and E.-h. Zahzah, "Online stochastic tensor decomposition for background subtraction in multispectral video sequences," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 106–113, 2015. 45

[178] C. Lu, J. Feng, Y. Chen, W. Liu, Z. Lin, and S. Yan, "Tensor robust principal component analysis with a new tensor nuclear

norm," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 925–938, 2019. 45

[179] D. Driggs, S. Becker, and J. Boyd-Graber, "Tensor robust principal component analysis: Better recovery with atomic norm regularization," *arXiv preprint arXiv:1901.10991*, 2019. 45

[180] A. J. Schofield, P. Mehta, and T. J. Stonham, "A system for counting people in video images using neural networks to identify the background scene," *Pattern Recognition*, vol. 29, no. 8, pp. 1421–1428, 1996. 45

[181] A. Tavakkoli, "Foreground-background segmentation in video sequences using neural networks," *Intelligent Systems: Neural Networks and Applications*, 2005. 45

[182] L. Maddalena and A. Petrosino, "A self-organizing approach to detection of moving patterns for real-time applications," in *International Symposium on Brain, Vision, and Artificial Intelligence*, pp. 181–190, Springer, 2007. 46

[183] L. Maddalena and A. Petrosino, "A self-organizing neural system for background and foreground modeling," in *International Conference on Artificial Neural Networks*, pp. 652–661, Springer, 2008. 46

[184] L. Maddalena and A. Petrosino, "Neural model-based segmentation of image motion," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 57–64, Springer, 2008. 46

[185] L. Maddalena and A. Petrosino, "A self-organizing approach to background subtraction for visual surveillance applications," *IEEE Transactions on Image Processing*, vol. 17, no. 7, pp. 1168–1177, 2008. 46

[186] L. Maddalena and A. Petrosino, "Multivalued background/foreground separation for moving object detection," in *International Workshop on Fuzzy Logic and Applications*, pp. 263–270, Springer, 2009. 46

[187] L. Maddalena and A. Petrosino, "A fuzzy spatial coherence-based approach to background/foreground separation for moving object detection," *Neural Computing and Applications*, vol. 19, no. 2, pp. 179–186, 2010. 46

[188] L. Maddalena and A. Petrosino, "The sobs algorithm: What are the limits?," in *2012 IEEE computer society conference on computer vision and pattern recognition workshops*, pp. 21–26, IEEE, 2012. 46

[189] L. Maddalena and A. Petrosino, "The 3dsobs+ algorithm for moving object detection," *Computer Vision and Image Understanding*, vol. 122, pp. 65–73, 2014. 46

[190] G. D. Sergio, V. P. Javier, *et al.*, "Simplified som-neural model for video segmentation of moving objects," in *2009 International Joint Conference on Neural Networks*, pp. 474–480, IEEE, 2009. 46

[191] M. I. Chacon-Murguía, G. Ramirez-Alonso, and S. Gonzalez-Duarte, "Improvement of a neural-fuzzy motion detection vision model for complex scenario conditions," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2013. 46

[192] G. Gemignani and A. Rozza, "A novel background subtraction approach based on multi layered self-organizing maps," in *2015 IEEE International Conference on Image Processing (ICIP)*, pp. 462–466, IEEE, 2015. 46

[193] L. Maddalena and A. Petrosino, "3d neural model-based stopped object detection," in *International Conference on Image Analysis and Processing*, pp. 585–593, Springer, 2009. 46

[194] L. Maddalena and A. Petrosino, "Self organizing and fuzzy modelling for parked vehicles detection," in *International con-*

ference on advanced concepts for intelligent vision systems, pp. 422–433, Springer, 2009. 46

[195] L. Maddalena and A. Petrosino, "Stopped object detection by learning foreground model in videos," *IEEE transactions on neural networks and learning systems*, vol. 24, no. 5, pp. 723–735, 2013. 46

[196] M. Babaee, D. T. Dinh, and G. Rigoll, "A deep convolutional neural network for background subtraction," *arXiv preprint arXiv:1702.01731*, 2017. 47

[197] Y. Wang, Z. Luo, and P.-M. Jodoin, "Interactive deep learning method for segmenting moving objects," *Pattern Recognition Letters*, vol. 96, pp. 66–75, 2017. 47

[198] S. Lee and D. Kim, "Background subtraction using the factored 3-way restricted boltzmann machines," *arXiv preprint arXiv:1802.01522*, 2018. 47

[199] T. P. Nguyen, C. C. Pham, S. V.-U. Ha, and J. W. Jeon, "Change detection by training a triplet network for motion feature extraction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 2, pp. 433–446, 2018. 47

[200] M. J. Shafiee, P. Siva, P. Fieguth, and A. Wong, "Real-time embedded motion detection via neural response mixture modeling," *Journal of Signal Processing Systems*, vol. 90, no. 6, pp. 931–946, 2018. 47

[201] R. Guo and H. Qi, "Partially-sparse restricted boltzmann machine for background modeling and subtraction," in *2013 12th International Conference on Machine Learning and Applications*, vol. 1, pp. 209–214, IEEE, 2013. 47

[202] L. Xu, Y. Li, Y. Wang, and E. Chen, "Temporally adaptive restricted boltzmann machine for background modeling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015. 47

[203] P. Xu, M. Ye, Q. Liu, X. Li, L. Pei, and J. Ding, "Motion detection via a couple of auto-encoder networks," in *2014 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, IEEE, 2014. 47

[204] P. Xu, M. Ye, X. Li, Q. Liu, Y. Yang, and J. Ding, "Dynamic background learning through deep auto-encoder networks," in *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 107–116, 2014. 47

[205] Z. Qu, S. Yu, and M. Fu, "Motion background modeling based on context-encoder," in *2016 Third International Conference on Artificial Intelligence and Pattern Recognition (AIPR)*, pp. 1–5, IEEE, 2016. 47

[206] Y. Zhang, X. Li, Z. Zhang, F. Wu, and L. Zhao, "Deep learning driven blockwise moving object detection with binary scene modeling," *Neurocomputing*, vol. 168, pp. 454–463, 2015. 47

[207] M. J. Shafiee, P. Siva, P. Fieguth, and A. Wong, "Embedded motion detection via neural response mixture background modeling," in *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 837–844, IEEE, 2016. 48

[208] M. Yazdi and T. Bouwmans, "New trends on moving object detection in video images captured by a moving camera: A survey," *Computer Science Review*, vol. 28, pp. 157–177, 2018. 48, 49, 51, 53, 54, 57

[209] Y. Sheikh, O. Javed, and T. Kanade, "Background subtraction for freely moving cameras," in *2009 IEEE 12th International Conference on Computer Vision*, pp. 1219–1225, IEEE, 2009. 49

[210] T. Brox and J. Malik, "Object segmentation by long term analysis of point trajectories," in *European conference on computer vision*, pp. 282–295, Springer, 2010. 50

[211] X. Yin, B. Wang, W. Li, Y. Liu, and M. Zhang, "Background subtraction for moving cameras based on trajectory-controlled segmentation and label inference," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 9, no. 10, pp. 4092–4107, 2015. 50

[212] S. Singh, C. Arora, and C. Jawahar, "Trajectory aligned features for first person action recognition," *Pattern Recognition*, vol. 62, pp. 45–55, 2017. 50

[213] S. Zhang, J.-B. Huang, J. Lim, Y. Gong, J. Wang, N. Ahuja, and M.-H. Yang, "Tracking persons-of-interest via unsupervised representation adaptation," *International Journal of Computer Vision*, vol. 128, no. 1, pp. 96–120, 2020. 50, 51

[214] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M.-M. Cheng, S. L. Hicks, and P. H. Torr, "Struck: Structured output tracking with kernels," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 2096–2109, 2015. 52

[215] W.-Z. Zhang, J.-G. Ji, Z.-Z. Jing, W.-F. Jing, and Y. Zhang, "Adaptive real-time compressive tracking," in *2015 Interna-*

tional Conference on Network and Information Systems for Computers, pp. 236–240, IEEE, 2015. 52

[216] M. Danelljan, F. Shahbaz Khan, M. Felsberg, and J. Van de Weijer, "Adaptive color attributes for real-time visual tracking," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1090–1097, 2014. 52, 61

[217] D. Du, H. Qi, L. Wen, Q. Tian, Q. Huang, and S. Lyu, "Geometric hypergraph learning for visual tracking," *IEEE transactions on cybernetics*, vol. 47, no. 12, pp. 4182–4195, 2016. 52

[218] T. Bouwmans, C. Silva, C. Marghes, M. S. Zitouni, H. Bhaskar, and C. Frelicot, "On the role and the importance of features for background modeling and foreground detection," *Computer Science Review*, vol. 28, pp. 26–91, 2018. 52

[219] Q. Zhao, Z. Yang, and H. Tao, "Differential earth mover's distance with its applications to visual tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 2, pp. 274–287, 2008. 52

[220] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference*

on computer vision and pattern recognition (CVPR'05), vol. 1, pp. 886–893, Ieee, 2005. 52

[221] C. Leng, H. Zhang, B. Li, G. Cai, Z. Pei, and L. He, "Local feature descriptor for image matching: A survey," *IEEE Access*, vol. 7, pp. 6424–6434, 2018. 52

[222] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005. 52

[223] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, pp. 404–417, Springer, 2006. 52

[224] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004. 53

[225] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*, pp. 778–792, Springer, 2010. 53

[226] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb:

An efficient alternative to sift or surf," in *2011 International conference on computer vision*, pp. 2564–2571, Ieee, 2011. 53

[227] S.-W. Ha and Y.-H. Moon, "Multiple object tracking using sift features and location matching," *International Journal of Smart Home*, vol. 5, no. 4, pp. 17–26, 2011. 53

[228] C. Kim, F. Li, A. Ciptadi, and J. M. Rehg, "Multiple hypothesis tracking revisited," in *Proceedings of the IEEE international conference on computer vision*, pp. 4696–4704, 2015. 53

[229] L. Leal-Taixe, C. Canton-Ferrer, and K. Schindler, "Learning by tracking: Siamese cnn for robust target association," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 33–40, 2016. 53, 54

[230] J. Ning, L. Zhang, D. Zhang, and C. Wu, "Robust object tracking using joint color-texture histogram," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 07, pp. 1245–1263, 2009. 54

[231] J. Pan, B. Hu, and J. Q. Zhang, "Robust and accurate object tracking under various types of occlusions," *IEEE transactions on circuits and systems for video technology*, vol. 18, no. 2, pp. 223–236, 2008. 54

[232] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 25, no. 5, pp. 564–577, 2003. 54

[233] B. Babenko, M.-H. Yang, and S. Belongie, "Robust object tracking with online multiple instance learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 8, pp. 1619–1632, 2010. 54

[234] M. A. Bagherzadeh and M. Yazdi, "Regularized least-square object tracking based on l2, 1 minimization," in *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*, pp. 635–639, IEEE, 2015. 54

[235] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool, "Robust tracking-by-detection using a detector confidence particle filter," in *2009 IEEE 12th International Conference on Computer Vision*, pp. 1515–1522, IEEE, 2009. 55

[236] A. Milan, K. Schindler, and S. Roth, "Multi-target tracking by discrete-continuous energy minimization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 2054–2068, 2015. 55

[237] N. Le, A. Heili, and J.-M. Odobez, "Long-term time-sensitive costs for crf-based tracking by detection," in *European Conference on Computer Vision*, pp. 43–51, Springer, 2016. 55

[238] J. Chen, H. Sheng, Y. Zhang, and Z. Xiong, "Enhancing detection model for multiple hypothesis tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 18–27, 2017. 55

[239] A. Sadeghian, A. Alahi, and S. Savarese, "Tracking the untrackable: Learning to track multiple cues with long-term dependencies," in *Proceedings of the IEEE international conference on computer vision*, pp. 300–311, 2017. 55

[240] J. Kuen, K. M. Lim, and C. P. Lee, "Self-taught learning of a deep invariant representation for visual tracking via temporal slowness principle," *Pattern recognition*, vol. 48, no. 10, pp. 2964–2982, 2015. 55

[241] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang, "Hierarchical convolutional features for visual tracking," in *Proceedings of the IEEE international conference on computer vision*, pp. 3074–3082, 2015. 55

[242] H. Li, Y. Li, F. Porikli, *et al.*, "Deeptrack: Learning discrim-

inative feature representations by convolutional neural networks for visual tracking.," in *BMVC*, vol. 1, p. 3, 2014. 55

[243] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4293–4302, 2016. 55

[244] N. Wang and D.-Y. Yeung, "Learning a deep compact image representation for visual tracking," *Advances in neural information processing systems*, vol. 26, 2013. 55

[245] N. Wang, S. Li, A. Gupta, and D.-Y. Yeung, "Transferring rich feature hierarchies for robust visual tracking," *arXiv preprint arXiv:1501.04587*, 2015. 55

[246] L. Wang, T. Liu, G. Wang, K. L. Chan, and Q. Yang, "Video tracking using learned hierarchical features," *IEEE Transactions on Image Processing*, vol. 24, no. 4, pp. 1424–1435, 2015. 56

[247] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual tracking with fully convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 3119–3127, 2015. 56

[248] M. Zhai, L. Chen, G. Mori, and M. Javan Roshtkhari, "Deep learning of appearance models for online object tracking," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pp. 0–0, 2018. 56

[249] P. Li, D. Wang, L. Wang, and H. Lu, "Deep visual tracking: Review and experimental comparison," *Pattern Recognition*, vol. 76, pp. 323–338, 2018. 56

[250] Z. Yu, P. Machado, A. Zahid, A. M. Abdulghani, K. Dashtipour, H. Heidari, M. A. Imran, and Q. H. Abbasi, "Energy and performance trade-off optimization in heterogeneous computing via reinforcement learning," *Electronics*, vol. 9, no. 11, p. 1812, 2020. 57

[251] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, vol. 2, pp. 142–149, IEEE, 2000. 57

[252] P. KaewTrakulPong and R. Bowden, "A real time adaptive visual surveillance system for tracking low-resolution colour targets in dynamically changing scenes," *Image and Vision Computing*, vol. 21, no. 10, pp. 913–929, 2003. 58

[253] T. Yang, Q. Pan, J. Li, and S. Z. Li, "Real-time multiple objects tracking with occlusion handling in dynamic scenes," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 970–975, IEEE, 2005. 58

[254] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via on-line boosting.," in *Bmvc*, vol. 1, p. 6, Citeseer, 2006. 59

[255] P. Heinemann, M. Plagge, A. Treptow, and A. Zell, "Tracking dynamic objects in a robocup environment-the attempto tübingen robot soccer team," *RoboCup-2003: Robot Soccer World Cup VII, Lecture Notes in Computer Science (CD-Supplement). Springer Verlag*, 2003. 59

[256] R. E. Schapire, "Explaining adaboost," in *Empirical inference*, pp. 37–52, Springer, 2013. 59

[257] M. Shah, O. Javed, and K. Shafique, "Automated visual surveillance in realistic scenarios," *IEEE MultiMedia*, vol. 14, no. 1, pp. 30–39, 2007. 59

[258] C. Bibby and I. Reid, "Robust real-time visual tracking using pixel-wise posteriors," in *European Conference on Computer Vision*, pp. 831–844, Springer, 2008. 60

[259] K. Huang, L. Wang, T. Tan, and S. Maybank, "A real-time object detecting and tracking system for outdoor night surveillance," *Pattern Recognition*, vol. 41, no. 1, pp. 432–444, 2008. 60

[260] S.-X. Li, H.-X. Chang, and C.-F. Zhu, "Adaptive pyramid mean shift for global real-time visual tracking," *Image and Vision Computing*, vol. 28, no. 3, pp. 424–437, 2010. 61

[261] A. Agarwal and S. Suryavanshi, "Real-time* multiple object tracking (mot) for autonomous navigation," *Tech. Rep.*, 2017. 62

[262] S. Minaeian, J. Liu, and Y.-J. Son, "Effective and efficient detection of moving targets from a uav's camera," *IEEE transactions on intelligent transportation systems*, vol. 19, no. 2, pp. 497–506, 2018. 62

[263] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990. 62

[264] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017. 63

[265] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014. 64, 65

[266] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1947–1950, IEEE, 2010. 64, 65

[267] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014. 64, 65

[268] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. 64, 65

[269] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Loihi:

A neuromorphic manycore processor with on-chip learning," *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018. 64, 65

[270] P. Lichtsteiner and T. Delbruck, "A 64x64 aer logarithmic temporal derivative silicon retina," in *Research in Microelectronics and Electronics, 2005 PhD*, vol. 2, pp. 202–205, IEEE, 2005. 65, 66

[271] L. Farian, J. A. Lenero-Bardallo, and P. Hafliger, "A bio-inspired aer temporal tri-color differentiator pixel array," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 9, no. 5, pp. 686–698, 2015. 65

[272] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014. 65

[273] N. Kasabov, K. Dhoble, N. Nuntalid, and G. Indiveri, "Dynamic evolving spiking neural networks for on-line spatio- and spectro-temporal pattern recognition," *Neural Networks*, vol. 41, pp. 188–201, 2013. 65, 66

[274] Z. Jiang, Z. Bing, K. Huang, and A. Knoll, "Retina-based pipe-like object tracking implemented through spiking neural

network on a snake robot," *Frontiers in neurorobotics*, vol. 13, p. 29, 2019. 66, 67, 68

[275] V. Oudjail and J. Martinet, "Bio-inspired event-based motion analysis with spiking neural networks.," in *VISIGRAPP (4: VISAPP)*, pp. 389–394, 2019. 66, 67

[276] R. Kuriyama, C. Casellato, E. D'Angelo, and T. Yamazaki, "Real-time simulation of a cerebellar scaffold model on graphics processing units," *Frontiers in cellular neuroscience*, vol. 15, p. 106, 2021. 68

[277] E. D'Angelo, S. Solinas, J. Mapelli, D. Gandolfi, L. Mapelli, and F. Prestori, "The cerebellar golgi cell and spatiotemporal organization of granular layer activity," *Frontiers in neural circuits*, vol. 7, p. 93, 2013. 68

[278] X. She, S. Dash, D. Kim, and S. Mukhopadhyay, "A heterogeneous spiking neural network for unsupervised learning of spatiotemporal patterns," *Frontiers in Neuroscience*, vol. 14, p. 1406, 2021. 69

[279] C. M. Parameshwara, S. Li, C. Fermüller, N. J. Sanket, M. S. Evanusa, and Y. Aloimonos, "Spikems: Deep spiking neural network for motion segmentation," in *2021 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), pp. 3414–3420, IEEE, 2021. 69

[280] T. Chen and S. Lu, "Object-level motion detection from moving cameras," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2333–2343, 2016. 70

[281] S. Yang, X. Hao, B. Deng, X. Wei, H. Li, and J. Wang, "A survey of brain-inspired artificial intelligence and its engineering," *Life Research*, vol. 1, no. 1, pp. 23–29, 2018. 70

[282] A. Podobas and S. Matsuoka, "Designing and accelerating spiking neural networks using opencl for fpgas," in *2017 International Conference on Field Programmable Technology (ICFPT)*, pp. 255–258, IEEE, 2017. 70, 72

[283] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239–255, 2010. 70

[284] G. Li, V. Talebi, A. Yoonessi, and C. L. Baker Jr, "A fpga real-time model of single and multiple visual cortex neurons," *Journal of neuroscience methods*, vol. 193, no. 1, pp. 62–66, 2010. 70

[285] J. Li, Y. Katori, and T. Kohno, "An fpga-based silicon neu-

ronal network with selectable excitability silicon neurons," *Frontiers in neuroscience*, vol. 6, p. 183, 2012. 70, 71

[286] A. S. Cassidy, J. Georgiou, and A. G. Andreou, "Design of silicon brains in the nano-cmos era: Spiking neurons, learning synapses and neural architecture optimization," *Neural Networks*, vol. 45, pp. 4–26, 2013. 71

[287] D. P. Moeys, T. Delbrück, A. Rios-Navarro, and A. Linares-Barranco, "Retinal ganglion cell software and fpga model implementation for object detection and tracking," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1434–1437, IEEE, 2016. 71

[288] Q. Chen, J. Wang, S. Yang, Y. Qin, B. Deng, and X. Wei, "A real-time fpga implementation of a biologically inspired central pattern generator network," *Neurocomputing*, vol. 244, pp. 63–80, 2017. 71

[289] K. Cheung, S. R. Schultz, and W. Luk, "Neuroflow: a general purpose spiking neural network simulation platform using customizable processors," *Frontiers in neuroscience*, vol. 9, p. 516, 2016. 72

[290] V. Sakellariou and V. Paliouras, "An fpga accelerator for spiking neural network simulation and training," in *2021 IEEE*

International Symposium on Circuits and Systems (ISCAS), pp. 1–5, IEEE, 2021. 72

[291] M. Im and S. I. Fried, "Directionally selective retinal ganglion cells suppress luminance responses during natural viewing," *Scientific reports*, vol. 6, no. 1, pp. 1–9, 2016. 78

[292] J. N. Kay, I. De la Huerta, I.-J. Kim, Y. Zhang, M. Yamagata, M. W. Chu, M. Meister, and J. R. Sanes, "Retinal ganglion cells with distinct directional preferences differ in molecular identity, structure, and central projections," *Journal of Neuroscience*, vol. 31, no. 21, pp. 7753–7762, 2011. 78

[293] P. Machado, A. Oikonomou, G. Cosma, and T. M. McGinnity, "Bio-inspired ganglion cell models for detecting horizontal and vertical movements," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2018. 78, 112, 189

[294] G. Deng and L. Cahill, "An adaptive gaussian filter for noise reduction and edge detection," in *1993 IEEE Conference Record Nuclear Science Symposium and Medical Imaging Conference*, pp. 1615–1619 vol.3, 1993. 84, 123

[295] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "Dl-resume: A delay learning-based remote supervised method for

spiking neurons," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 12, pp. 3137–3149, 2015. 86

[296] OpenCV, "OpenCV: Color conversions," 2021. Available onine, https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html, last accessed: 30/07/2021. 93

[297] A. Kessy, A. Lewin, and K. Strimmer, "Optimal whitening and decorrelation," *The American Statistician*, vol. 72, no. 4, pp. 309–314, 2018. 94

[298] Brian, "Introduction to Brian 2," 2021. Available onine, https://brian2.readthedocs.io/en/2.0rc/resources/tutorials/2-intro-to-brian-synapses.html, last accessed: 30/07/2021. 97, 124

[299] L. Maddalena and A. Petrosino, "Background subtraction for moving object detection in rgbd data: A survey," *Journal of Imaging*, vol. 4, no. 5, p. 71, 2018. 115

[300] D. Holmes, "Reconstructing the retina.," *Nature*, vol. 561, no. 7721, pp. S2–S3, 2018. 115

[301] A. Stalin and W. Amitabh, "Bsfd: Background subtraction frame difference algorithm for moving object," *Journal of The-*

oretical and Applied Information Technology, vol. 60, no. 3, pp. 623–628, 2014. 115

[302] A. Vacavant, T. Chateau, A. Wilhelm, and L. Lequievre, "A benchmark dataset for outdoor foreground/background extraction," in *Asian Conference on Computer Vision*, pp. 291–300, Springer, 2012. 115

[303] R. D. Sharma, S. L. Agrwal, S. K. Gupta, and A. Prajapati, "Optimized dynamic background subtraction technique for moving object detection and tracking," in *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*, pp. 1–3, IEEE, 2017. 115

[304] M. Rashid and V. Thomas, "A background foreground competitive model for background subtraction in dynamic background," *Procedia technology*, vol. 25, pp. 536–543, 2016. 115

[305] J.-W. Seo and S. D. Kim, "Dynamic background subtraction via sparse representation of dynamic textures in a low-dimensional subspace," *Signal, Image and Video Processing*, vol. 10, no. 1, pp. 29–36, 2016. 115

[306] O. Community, "OpenCV," 2020. Available online, https://opencv.org/, last accessed: 2020-01-22. 117, 124, 129

[307] F. Standard C++, "STL C++17," 2017. Available online, https://isocpp.org/std/status, last accessed: 23/11/2020. 124

[308] Boost, "Boost C++ libraries," 2020. Available online, http://www.boost.org/, last accessed: 23/11/2020. 124

[309] R. Jolivet, T. J. Lewis, and W. Gerstner, "Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accuracy," *Journal of neurophysiology*, vol. 92, no. 2, pp. 959–976, 2004. 125

[310] R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *Journal of neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005. 125

[311] L. A Pastur-Romay, A. B Porto-Pazos, F. Cedrón, and A. Pazos, "Parallel computing for brain simulation," *Current Topics in Medicinal Chemistry*, vol. 17, no. 14, pp. 1646–1668, 2017. 145

[312] R. Brooks, D. Hassabis, D. Bray, and A. Shashua, "Is the brain a good model for machine intelligence?," *Nature*, vol. 482, no. 7386, p. 462, 2012. 145

[313] S. Herculano-Houzel, "The human brain in numbers: a linearly scaled-up primate brain," *Frontiers in human neuroscience*, p. 31, 2009. 145

[314] P. J. Fox, "Massively Parallel Neural Computation," *University of Cambridge Computer Laboratory*, no. 830, pp. 1–105, 2013. 145

[315] Intel, "What Is a GPU? Graphics Processing Units Defined," 2020. Available online, https://www.intel.co.uk/content/www/uk/en/products/docs/processors/what-is-a-gpu.html, last accessed: 2021-03-10. 146, 149

[316] H. S. de Andrade, *Software Concerns for Execution on Heterogeneous Platforms*. PhD thesis, Chalmers University of Technology, 2018. 148

[317] CodeProjects, "Part 5: OpenCL Buffers and Memory Affinity - CodeProject," 2011. Available online, https://www.codeproject.com/Articles/201258/Part-5-OpenCL-Buffers-and-Memory-Affinity, last accessed: 2021-04-02. 154

[318] P. Machado, A. Oikonomou, J. F. Ferreira, and T. McGinnity, "Hsmd: An object motion detection algorithm using a hybrid

spiking neural network architecture," *IEEE Access*, pp. 1–1, 2021. Available onine, https://doi.org/10.1109/ACCESS.2021.3111005, last accessed: 2021-07-31. 189