NATIONAL TECHNICAL UNIVERSITY OF ATHENS
POSTGRADUATE PROGRAM ON AUTOMATION SYSTEMS
SCHOOL OF MECHANICAL ENGINEERING

Multicopter control using dynamic vision and neuromorphic computing

THESIS PROJECT

Evangelos Ntouros

Supervisor: Kostas J. Kyriakopoulos
Professor NTUA

CONTROL SYSTEMS LAB
Athens, October 2022

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΔΠΜΣ ΣΥΣΤΗΜΑΤΑ ΑΥΤΟΜΑΤΙΣΜΟΥ
ΣΧΟΛΗ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

Έλεγχος πολυκοπτέρου με χρήση δυναμικής όρασης και νευρομορφικού υπολογιστικού συστήματος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
του
Ευάγγελου Ντούρου

Επιβλέπων: Κώστας Ι. Κυριακόπουλος
Καθηγητής Ε.Μ.Π.

ΕΡΓΑΣΤΗΡΙΟ ΑΥΤΟΜΑΤΟΥ ΕΛΕΓΧΟΥ
Αθήνα, Οκτώβριος 2022

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΔΠΜΣ ΣΥΣΤΗΜΑΤΑ ΑΥΤΟΜΑΤΙΣΜΟΥ
ΣΧΟΛΗ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

Έλεγχος πολυκοπτέρου με χρήση δυναμικής όρασης και νευρομορφικού υπολογιστικού συστήματος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
του
Ευάγγελου Ντούρου

Επιβλέπων: Κώστας Ι. Κυριακόπουλος
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε απο την τριμελή επιτροπή την 17η Οκτωβρίου, 2022.

.....	.....	.....
Κώστας Ι. Κυριακόπουλος	Ευάγγελος Γ. Παπαδόπουλος	Κωνσταντίνος Τζαφέστας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

ΕΡΓΑΣΤΗΡΙΟ ΑΥΤΟΜΑΤΟΥ ΕΛΕΓΧΟΥ
Αθήνα, Οκτώβριος 2022

.....
Ευάγγελος Ντούρος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός
και Μηχανικός Υπολογιστών Ε.Μ.Π.

στην οικογένειά μου

Contents

Περίληψη

Σε αυτήν εργασία, αναπτύσεται μια ολοκληρωμένη στρατηγική ελέγχου για μη επανδρωμένα πολυκόπτερα με σκοπό την αποτελεσματική παρακολούθηση στόχων που ορίζονται από συνεχή τμήματα περιγεγραμμένων επιφανειών σε ένα περιβάλλον, όπως πεζοδρόμια. Το ρομπότ που χρησιμοποιήθηκε φέρει ένα δυναμικό αισθητήρα όρασης (DVS), ένα νευρομορφικό υπολογιστικό σύστημα (SpiNN-3) καθώς και ένα συμβατικό (NVIDIA Jetson AGX Xavier). Η DVS παράγει events, τα οποία τροφοδοτούνται ασύγχρονα σε μία υλοποίηση του μετασχηματισμού Hough με τη χρήση των Spiking Neural Networks (SNN) ώστε να εντοπίζει τις γραμμές του περιβάλλοντος. Η έξοδος του εν λόγω νευρωνικού δρομολογείται σε έναν αλγόριθμο παρακολούθησης που τρέχει στο Jetson AGX Xavier και σκοπός του έιναι να εντοπίζει και να παρακολουθεί τη γραμμή που αντιστοιχεί στο πεζοδρόμιο. Στη συνέχεια, αυτή αποτελεί την είσοδο ενός ελεγκτή οπτικής ανατροφοδότητης με βάση την εικόνα (IBVS), ο οποίος οδηγεί αυτόνομα το ρομπότ κατα μήκος του στόχου. Η προτεινόμενη αρχιτεκτονική πετυχαίνει υψηλής ακρίβειας εντοπισμό του στόχου, με πολύ γρήγορη ασύγχρονη ανανέωσή του. Ταυτόχρονα επιτυγχάνεται μείωση της καταναλισκόμενης, απο το σύστημα, ισχύος. Για την επίδειξη της αποτελεσματικότητας του προτεινόμενου συνολικού σχήματος ελέγχου, χρησιμοποιήθηκε ένα πολυκόπτερο οκτώ ελίκων με την DVS και το SpiNN-3, και πραγματοποιήθηκε μία σειρά πειραμάτων.

Λέξεις Κλειδιά

UAV, event-based vision, DVS, neuromorphic computing, SpiNNaker, FPGA, VLSI design, SNN, Hough Transform, IBVS control

Abstract

In this thesis, an end-to-end control strategy for the efficient tracking of contour-based areas, such as pavements, is presented using a multirotor Unmanned Aerial Vehicle (UAV). The robot is equipped with a bio-inspired Dynamic Vision Sensor (DVS), a Neuromorphic (SpiNN-3), and a conventional (NVIDIA Jetson AGX Xavier) Computing platform. Concerning the detection part, a hybrid framework is developed which combines conventional and neuromorphic algorithms and hardware to detect and track contour-based areas. The DVS generates events, which are asynchronously fed into a Neuromorphic Hough Transform algorithm running on the SpiNN-3 board and implemented as a Spiking Neural Network (SNN) to detect lines in the visual scene. The SNN output is fed into a tracking algorithm, running in Jetson AGX Xavier, which tracks the pavement. Next, the asynchronous output of the detection module is fed into an IBVS controller, which allows the multirotor to autonomously navigate along the detected contour. The proposed architecture achieves high precision detection with fast asynchronous update while optimizing the system in terms of power consumption. A set of real-time experiments in various settings employing an octocopter equipped with a downward-looking DVS and a SpiNN-3 board demonstrate the effectiveness of the suggested control scheme.

Key Words

UAV, event-based vision, DVS, neuromorphic computing, SpiNNaker, FPGA, VLSI design, SNN, Hough Transform, IBVS control

Ευχαριστίες

Με την ολοκλήρωση της παρούσας πτυχιακής εργασίας, θα ήθελα να ευχαριστήσω όλους όσους με βοήθησαν κατα τη διάρκειά της, συντελώντας καθοριστικά στην εκπόνησή της.

Ευχαριστώ θερμά τον καθηγητή Κώστα Ι. Κυριακόπουλο για την ευκαιρία που μου έδωσε αναθέτοντάς μου το πολύ ενδιαφέρον θέμα αυτό και για την καθοδήγηση του καθώς και τον καθηγητή Γεώργιο Καρρά για την πολυεπίπεδη στήριξή του στα επιστημονικά και ερευνητικά θέματα που προέκυψαν κατα τη διάρκεια αυτής της προσπάθειας.

Ιδιαιτέρως θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα Σωτήρη Ασπράγκαθο για τη βοήθειά του στην εκτέλεση των πειραμάτων καθώς και για τη συνεργασία μας στη συγγραφή της επιστημονικής δημοσίευσης, βάση της οποίας αποτέλεσε η παρούσα εργασία.

Τέλος, θα ήθελα να ευχαριστήσω πολύ την οικογένεια μου και τους κοντινούς μου ανθρώπους των οποίων η στήριξη έπαιξε καταλυτικό ρόλο στην ολοκλήρωση της εργασίας αυτής.

Acknowledgment

With the completion of this thesis, I would like to thank everyone who helped me throughout this effort.

Firstly, I would like to thank professor Kostas J. Kyriakopoulos for providing me with the opportunity to investigate such an interesting topic as well as for his guidance during this time. I would also like to thank professor Georgios Karras for providing me with insightful directions and approaches to various scientific/research issues that occurred.

I want to acknowledge Ph.D. candidate Sotirios Aspragkathos, too, and thank him for his support with the experiments of this thesis as well as for our cooperation in writing the scientific paper that derived from this thesis's work.

Finally, I would like to thank my family and the closest people around me, whose continuous support and encouragement was a crucial factor in the completion of this work.

Acronyms

A

AER Adress-Event Representation. 21

ANNs Artificial Neural Networks. 23

C

CNNs Convolutional Neural Networks. 19

CPUs Central Processing Units. 19

D

DoF Degree of Freedom. 20

DR Dynamic Range. 21

DVS Dynamic Vision Sensor. 19

E

EoP End of Packet. 27

eps events per second. 26

F

FoV Field of View. 22

FPGAs Field Programmable Gate Arrays. 21

G

GPIO General Purpose Input Output. 29

I

IBVS Image-Based Visual Servo. 20

IMSE Instituto de Microelectronica de Sevilla. 22

L

LIF Leaky Integrate and Fire. 23

LVCMOS Low Voltage Complementary Metal Oxide Semiconductor. 27

M

MAC Multiply And Accumulate. 23

MPC Model Predictive Control. 47

N

NoC Network on Chip. 24

P

PBVS Position-Based Visual Servo. 41

R

ROI Region of Interest. 34

ROS Robot Operating System. 20

RTL Register Transfer Level. 29

S

SATA Serial AT Attachment. 24

SNNs Spiking Neural Networks. 19

U

UAVs Unmanned Aerial Vehicles. 19

UDP User Datagram Protocol. 24

V

VIO Visual-Inertial Odometry. 19

VLSI Very Large Scale Integration. 21

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Multirotor Unmanned Aerial Vehicles (UAVs), or multicopters, are already used in an increasing range of civil applications, from infrastructure inspections to aerial photography and videography. They can maneuver around barriers, fly at various speeds, retain their position, hover over targets, and operate well in both indoor and outdoor environments. Their flight time and overall endurance have significantly increased as a result of recent developments in navigation and perception sensors, placing them among the best platforms for activities requiring visual surveillance and area coverage.

Recent technological advancements have led to the creation of a variety of bio-inspired vision sensors, such as the Dynamic Vision Sensor (DVS). Its operation is inspired by the biological retina and it broadcasts the visual scene as a sequence of asynchronous events generated by brightness changes. DVSs are characterized by high speed, low latency, and excellent dynamic range, which allow them to exhibit superior performance in challenging computer vision and robotic tasks where conventional (frame-based) cameras may show substandard performance.

To fully utilize Dynamic Vision Sensors, new processing techniques are required to handle the asynchronous output [1]. A suitable choice is Spiking Neural Networks (SNNs), that use a bio-inspired model of neurons to carry out the computation. SNNs are by their nature in line with the operation of DVSs since each neuron in the input layer can be implemented to represent a pixel of the DVS. Additionally, SNNs use a lot less computational resources than standard deep Convolutional Neural Networks (CNNs), resulting in reduced power consumption and reaction latency, which are required in autonomous robotic systems and especially UAVs.

Furthermore, recent advancements in the development of specialized spiking neural hardware platforms for computing and sensing have been introduced, such as TrueNorth [2], Neurogrid [3], SpiNNaker [4, 5, 6] and Loihi [7], that simulate the behaviour of such networks directly in hardware. These platforms provide outstanding performance and energy efficiency during the training and inference stage, however, they are less adaptable than general-purpose Central Processing Units (CPUs).

Various studies can be found in the literature that exploit the benefits of DVSs, but very few combine them with the advantages of neuromorphic hardware for asynchronous data acquisition and processing. Instead they utilize conventional processor units. While these approaches result in overall low latency systems, they still consume a lot of processing power. Examples can be found in feature detection [8], feature tracking [9, 10], clustering [11], pose tracking [12] and Visual-Inertial Odometry (VIO) [13]. In [14], event frames were used to close the loop during the autonomous flight of a quadrotor vulnerable to failures as well as estimate the position and yaw orientation of the vehicle using VIO. A method for onboard object tracking and recognition for a UAV was reported in [15]. An autonomous UAV landing method based on the optical flow of event frames collected from a downward-looking DVS was described in [16]. In [17] a new UAV dodging technique was presented, where to execute evasive maneuvering, the

authors proposed a potential field-based technique that relied on the spatiotemporal continuity of events to find and follow moving objects. The research described in [18] proposed a method for directing an ornithopter robot towards a target by employing event-based vision techniques.

There are studies, however, that utilize neuromorphic computing. For example, in the case of [19], a DVS and an SNN running on a SpiNNaker platform are combined with a servo motor for a goalkeeper application to intercept the incoming ball. A spiking neural network is utilized exploiting the capabilities of Loihi neuromorphic platform to land a micro air vehicle. However, a conventional frame-based camera [20] is employed in this situation. Finally, [21] examines the use of an SNN on a neuromorphic platform to create an event-based vision algorithm in a drone controller, but only for 1 Degree of Freedom (DoF).

Thesis Contributions

The goal of this thesis is to fill in the gap in the aforementioned studies, developing an end-to-end event-based control strategy for contour-based areas that utilizes dynamic vision and neuromorphic computing in its perception module, to control a 6 DoF UAV in a visual surveillance task. This is accomplished in the following steps.

- Integration of a DVS and a neuromorphic platform on the hardware of the UAV, using a custom designed VLSI interface.
- Development of an asynchronous detection framework for contour-based areas which runs on a hybrid computing system (neuromorphic-conventional), receives DVS events as input, detects the contour-based area, and extracts the features for the control module.
- Development of an Image-Based Visual Servo (IBVS) controller to achieve error convergence close to zero in the surveillance task.

The perception algorithm is initially (in the scope of this thesis) developed and tested to detect pavements. Nevertheless, an enhanced set of contour-based areas (e.g. road lines, forest paths, power lines, coastlines etc.) can be detected given the appropriate modifications and the presence of training data.

The detection framework is developed under the Robot Operating System (ROS) [22], in C++, and the VLSI interface is designed using VHDL. All the code is available in Github (`https://github.com/ntouev`) under **ev_snn_percept/** and **spinn_aer_if/** repositories respectively.

1.2 Retinomorphic Event-Based Vision

Despite all the impressive progress made during the last decades, in the fields of information technology, microelectronics and computer science, artificial sensory and information processing systems are still much less effective in dealing with real-world tasks than their biological counterparts. Even small insects outperform the most powerful computers in routine functions involving real-time sensory data processing, perception tasks, and motor control with the minimum amount of energy.

State of the art image sensors suffer from limitations imposed by their frame-based nature. Information is conveyed at specific time intervals and this impacts on the temporal resolution of the sensor. This is diametrically opposed to real world scenarios, where changes happen asynchronously and fast. In addition to that, each frame carries a lot of redundancy, since the majority of the scenery pixels do not change at all. The problem deriving from these shortcomings for real-time control systems is the amount of effort (power consumption) needed to process all that information as fast as possible.

On the other hand, the biological retina (its structure is described in detail in [23]) converts spatio-temporal information contained in the incident light from the visual scene into spike trains. These are conveyed to the visual cortex by retinal ganglion cells, whose axons form the fibers of the optic nerve. The information carried by these spikes is maximized by the retinal processing, encompassing

highly evolved adapting filtering and sampling mechanisms to improve coding efficiency, such as local automatic gain control, bandpass spatio-temporal filtering, and rectification in ON and OFF output cell types. Furthermore, the varying distribution of different receptor types along with corresponding pathways across the retina combined with precise rapid eye movements elicit the illusion of high spatial and temporal resolution everywhere. In reality, though, the retina samples at high spatial but low temporal resolution in the center and the opposite in the periphery.

On the ground of the above, the field of Retinomorphic Event-Based Vision has attracted a lot of research interest over the last two decades. A set of sensors were developed, whose operation principles are inspired by biology, however the focus is less on a faithful reproduction of the biological ones, and more on devising engineering solutions to real-world vision problems.

1.2.1 Address Event Representation (AER)

Since the 1940s, an effort has been put, into building artificial biology-like neural networks and sensors, initially for biology research and afterward for exploiting the advantages in real-life engineering problems. Many technologies were tried, in that direction, and the most popular up to now remains the Very Large Scale Integration (VLSI) digital circuit design. The reason is most importantly the high parallelism VLSI can offer. However, there is a fundamental problem that arises. It is the number of interconnections between the various neurons. A biological retina, for example, sends information to the brain via the optical nerve, which is formed by the axons of millions of ganglion cells existing in the retina. Translating this situation to an artificial retina would imply that each pixel of an image sensor would have its own wire to convey its data.

Given the restrictions posed by chip interconnect and packaging technologies and that mainstream VSLI technology does not allow for dense 3-D wiring, this is obviously not a transferable approach. However, there is a workaround. Leveraging the five orders of magnitude or more of difference in bandwidth between a neuron (typically spiking at rates between 10 and 1000 Hz) and a digital bus enables engineers to replace thousands of dedicated point-to-point connections with a few metal wires and lots of switches. To do so, the traffic over these wires is time multiplexed using a packet-based or “event-based” data protocol called Adress-Event Representation (AER).

In an AER link, a transmitter module includes, e.g., an array of neurons generating spikes. Each neuron is assigned an address, such as its x,y coordinates within the array. Neurons generate spikes and are arbitrated and put on an intermodule high-speed asynchronous AER bus, implementing a time-multiplexing strategy where all computing elements (pixels, neurons, etc.) share the same physical bus to transmit their pulses, together with the implicit timing information. In this asynchronous protocol, temporal information is self-encoded in the timing of the events, and it is explicitly added to the address in the form of a timestamp only when processing takes place in “non-AER” processing units, such as Field Programmable Gate Arrays (FPGAs) or digital processors [24]. Finally, addresses are received, read, and decoded by the receiver module and sent to the corresponding destination neuron or neurons.

1.2.2 Dynamic Vision Sensor (DVS)

All conventional frame-based image sensors completely neglect the dynamic information immanent to natural scenes. In an attempt to realize a practical transient vision device, the DVS was developed [25, 26, 27]. This type of vision sensor is sensitive to the scene dynamics and directly responds to changes, i.e., temporal contrast, pixel individually, and near real-time response. The gain in terms of temporal resolution with respect to standard frame-based image sensors is dramatic. But also other performance parameters like the intrascene Dynamic Range (DR) greatly profit from the biological approach. This type of sensor is very well suited for a plethora of robotic applications such as high-speed motion detection and tracking.

The DVS pixel models a simplified three-layer retina (Fig. 1.1), implementing an abstraction of the information flow in the biological retina. The functionality of the equivalent electronic circuit, as well as the similarities and differences with the biological retina, are analyzed in detail in [23]. Single pixels are

spatially decoupled but take into account the temporal development of the local light intensity. The DVS pixel autonomously responds to relative changes in intensity at microsecond temporal resolution over six decades of illumination.

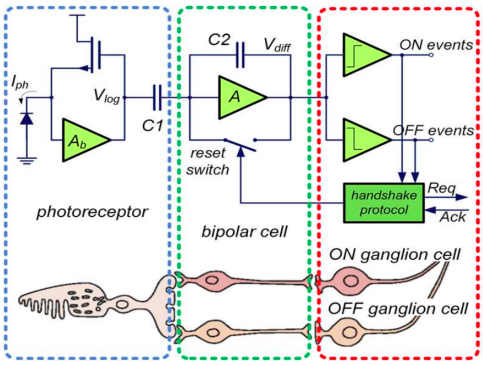


Figure 1.1: Modeling of the DVS in silicon [23].

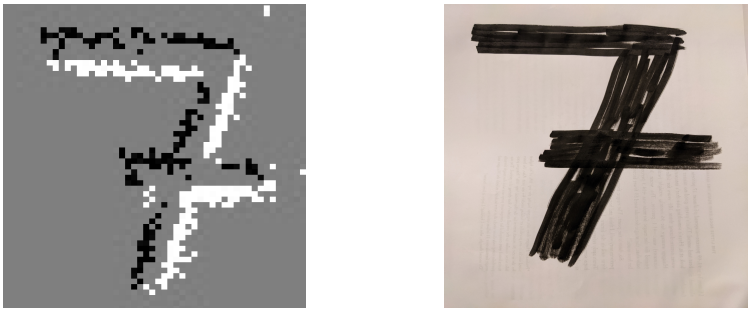
These properties are directly associated with abandoning the frame principle and modeling three key properties of biological vision: the sparse, event-based output; the representation of relative luminance change; and the rectification of positive and negative signals into separate output channels (ON/OFF). The major consequence of this bioinspired approach and most distinctive feature with respect to standard imaging is that the control over the acquisition of the visual information is no longer being imposed on the sensor in the form of an external clock, but the decision-making is transferred to the single pixel that handles its own visual information individually and autonomously.

Figure 1.2 depicts the output of the DVS by accumulating events for 50 $msec$ for a moving black-drawn number on a white paper. ON events are illustrated with white colour and OFF events with black. In this particular case, it can be extracted that the paper is moving upwards.

The DVS used in the setup of the UAV, shown in Figure 1.3, was provided by the Neuromorphic Group of the Instituto de Microelectronica de Sevilla (IMSE) [28]. It has a 128×128 resolution, while the camera lens is chosen, appropriately, to give a narrow Field of View (FoV) (approximatelyly $1.5m \times 1.5m$ at height $5m$), to address the issue of low resolution.

1.3 Neuromorphic Engineering

Neuromorphic engineering or computing is an interdisciplinary subject, developed in the late 1980s, that takes inspiration from biology, physics, mathematics, computer science, and electronic engineering to design artificial neural systems, such as sensors and computing systems, whose physical architecture and design principles are based on those of biological nervous systems. Apart from that, neuromorphic



(a) Accumulated events. (b) Black-drawn seven.

Figure 1.2: A black-drawn seven in white paper, moves upgrards in front of the DVS, generating OFF events in the leading **(black)** edges and ON events in the following **(white)** edges.

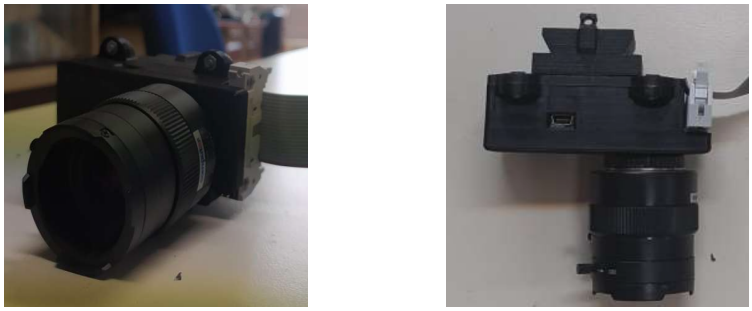


Figure 1.3: The DVS used in the setup of the UAV, from two angles.

engineering helps in understanding how the morphology of individual neurons and overall architectures creates desirable computations, affects how information is represented, influences robustness to damage, incorporates learning and development, adapts to local change (plasticity), and facilitates evolutionary change.

1.3.1 Spiking Neural Networks (SNNs)

Neuromorphic engineering uses dynamic models of neurons that approach the behaviour of biological neurons to build biological-like network architectures. These architectures are the Spiking Neural Networks (SNNs).

Leaky Integrate and Fire (LIF) Neuron

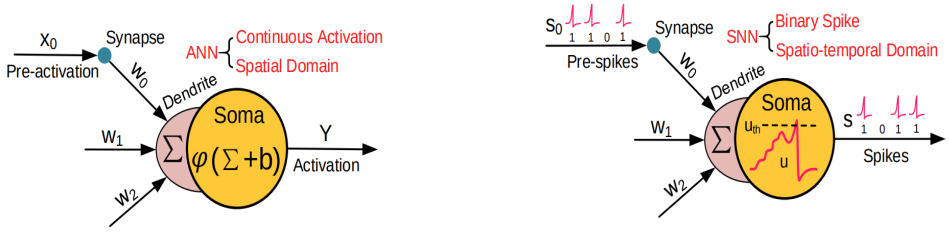
The most used neuron model is the Leaky Integrate and Fire (LIF) model. It represents the neuron as a parallel combination of a "leaky" resistor (conductance, g_L) and a capacitor (C). A current source $I(t)$ is used as the synaptic current input to charge up the capacitor to produce the membrane potential $V_m(t)$. When the potential exceeds a threshold value ($V_m(t) \geq V_{th}$), it increases rapidly producing the so-called spike. The neuron then resets, and V_m returns to a resting potential E_L after a small delay. During this time a voltage dip is observed (Fig. 1.5 between 50 and 60 ms) and any possible incoming spike will not affect the neuron's membrane potential at all. As soon as V_m has returned to E_L is "ready" to receive input again. In the absence of any incoming spikes, the membrane potential decays exponentially. Figure 1.5 depicts the aforementioned behaviour.

$$C\frac{dV_m(t)}{dt} = -g_L(-V_m(t) - E_L) + I(t) \tag{1.1}$$

Artificial vs Spiking Neuron

Figure 1.4a depicts the model of a typical artificial neuron, where x, y, w, and b are the input, output, synaptic weight, and bias, respectively of input neuron, and j is the index of input neuron. The $\Phi(\cdot)$ is a nonlinear activation function, e.g. $\Phi(x) = ReLU(x) = max(x, 0)$. Neurons in Artificial Neural Networks (ANNs) communicate with each other using activations coded in high-precision and continuous values and only propagate information in the spatial domain (i.e. layer by layer). The major operations executed by ANNs are the Multiply And Accumulate (MAC) of inputs and weights.

On the other hand, figure 1.4b shows a typical spiking neuron, which has a similar structure but different behavior compared to the artificial neuron. By contrast, spiking neurons communicate through spike trains coded in binary events rather than the continuous activations in ANNs. The *dendrites* integrate the input spikes and the *soma* consequently conducts nonlinear transformation to produce the output spike train [29]. SNNs represent information in spike patterns, and each spiking neuron experiences rich



(a) (b)

Figure 1.4: Models of an Artificial and a Spiking Neuron [29].

dynamic behavior. Specifically, besides the information propagation in the spatial domain, the current state is tightly affected by the past history in the temporal domain. Therefore, SNNs usually have more temporal versatility but lower precision compared to ANNs.

Power Efficiency

Since a spike only fires when the membrane potential exceeds a threshold, the entire spike signals are often sparse and the computation can be event-driven (only enabled when an input spike arrives). Furthermore, since spikes are binary, i.e. 0 or 1, the costly multiplication between the input and weight can be removed. Consequently, SNNs can achieve lower power consumption compared to ANNs.

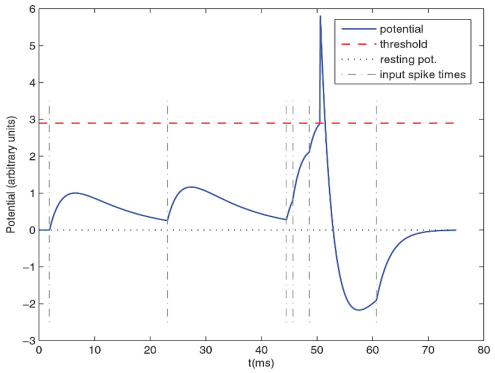


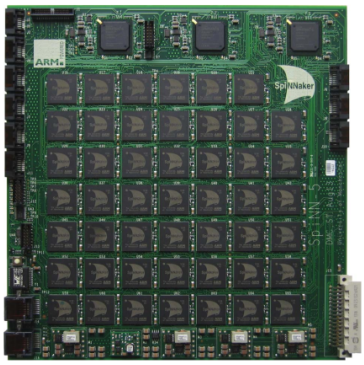
Figure 1.5: LIF neuron model response to incoming spikes. Y axis is the membrane potential $V_m(t)$ [30].

1.3.2 Neuromorphic Hardware

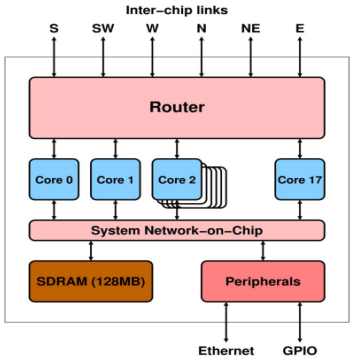
To fully exploit the advantages of SNNs, various special hardware platforms have been designed. They consist of analog, digital or mixed-mode analog/digital VLSI systems to help simulate SNNs directly on hardware. The final platform has the form of an accelerator where neurons' state and networks' interconnections are modeled and run fast and efficiently, in the absence of a CPU module.

SpiNNaker is a million-core computing engine (Fig. 1.6a) whose primary goal is to simulate the behaviour of up to a billion neurons in real-time [31]. It consists of many boards assembled, each one (Fig. 1.6b) combining 48 SpiNNaker nodes. The architecture of each node can be seen in Figure 1.6c and it incorporates a packet router, 18 ARM968 cores, a System Network on Chip (NoC), an SDRAM, and peripherals for I/O, while 16 of the 18 cores model up to 1000 neurons each one with 1000 available synapses, one serves as a master node for software purposes and the last one is a spare node to achieve a fault-tolerant system operation. Packets of 5 or 9 bytes, model the spikes of the neurons and are transmitted as User Datagram Protocol (UDP) packets in discrete timesteps of 1 *msec* inside the node utilizing the NoC system, from node to node inside the 48-node board via the packet routers and finally from board to board inside the whole engine using Serial AT Attachment (SATA) links.

(a) The Spinnaker Engine.



(b) The 48-node board.



(c) The architecture of a SpiNNaker node [6].

Figure 1.6: SpiNNaker system submodules.

The neuromorphic hardware, chosen for our setup (depicted in Figure 1.7), is the SpiNN-3 board [32], a small SpiNNaker platform suitable for mobile robot applications. It can simulate up to 16k neurons and provides the ability to inject real-time spikes (e.g. DVS events) using a special communication protocol. Communication with external computing platforms is achieved via an Ethernet port. The SpiNN-3 board was provided by the APT Group in the School of Computer Science of the University of Manchester.

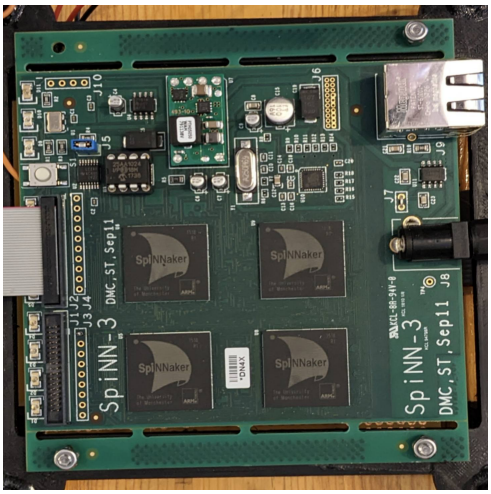


Figure 1.7: The SpiNN-3 board.

1.4 Document Structure

The rest of the document is organized as follows. The essential hardware design for interfacing the DVS to the SpiNN-3 board is analyzed in Chapter 2. In Chapter 3 the asynchronous contour-based area detection framework is developed and validated with the appropriate metric. Chapter 4 formulates the control strategy, and lastly, the experimental validation of the proposed method and the results are presented in Chapter 5, while Chapter 6 concludes the thesis.

Chapter 2

Interfacing DVS to SpiNNaker

2.1 AER Interface

The AER interface, described in 1.2.1, is an asynchronous interface that implements a 4-phase handshake protocol with active-low request and acknowledge signals. Data (events) are transmitted in parallel, 16-bit words. The logic levels 0 and 1 are represented by voltages of $0V$ and $5V$ or $3.3V$ respectively (depending on the device).

2.1.1 4-phase Handshake Protocol and Timings

The 4-phase handshake protocol, implemented by AER devices, requires both request and acknowledge signals to have an initial state of HIGH. After each transmission cycle ends, they should return to this state. The actions of the sender and receiver can be seen in Figure 2.1 and are summed up below.

- The sender adds the data to be transmitted on the bus and pulls request LOW.
- When ready, the receiver accepts the data and pulls acknowledge LOW.
- Next, the sender pulls request HIGH (initial state).
- Finally, the receiver pulls acknowledge HIGH (initial state).

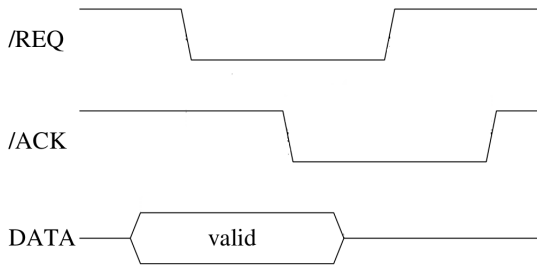


Figure 2.1: 4-phase handshake protocol signals.

According to [33], to determine the bus bandwidth, the size of the AER retina should be considered. The DVS has a 128×128 resolution, corresponding to a 14-bit address space, i.e. 16384 addresses. By taking the rational assumptions, that the maximum frequency of a neuron will not exceed $100Hz$ and that never more than half the pixels will be active, we can conclude that an AER bus, needs to transmit up to a maximum event rate of 819200 events per second (eps). This leads to a hard design constraint regarding the speed of the designed Interface, which has to handle transmissions up to $1Meps$, i.e. each transmission should last less than $1\mu s$.

2.2 SpiNNaker Link

The SpiNNaker Link interface [34] is an asynchronous interface that implements a 2-phase handshake protocol for data transmission. It is primarily designed for inter-chip communication in SpiNNaker boards. It comprises 2 independent unidirectional links for input and output. The data is conveyed in packet form, using a self-timed 2-of-7 protocol.

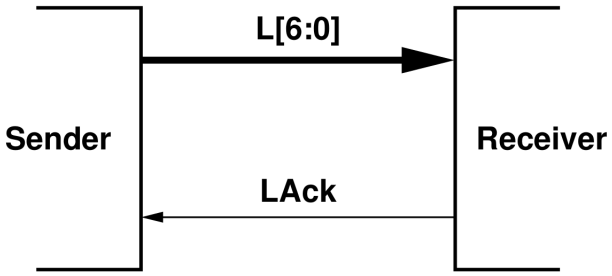


Figure 2.2: SpiNNaker Link overview.

2.2.1 2-phase Handshake Protocol

The 2-phase handshake protocol is a simple synchronization mechanism implemented between the sender and the receiver.

- The sender adds the data to be transmitted on the bus.
- When ready, the receiver accepts the data and toggles the acknowledge signal, read by the sender, to indicate that is available for a new transmission.

2.2.2 Self-timed 2-of-7 Coding

Each unidirectional link of the SpiNNaker Link consists of 7 data wires, L[6:0], and an acknowledge wire, LAck. The SpiNNaker chip I/O drivers use Low Voltage Complementary Metal Oxide Semiconductor (LVCMOS) voltages and thresholds, so HIGH and LOW are represented by $0V$ and $1.8V$, respectively. Following chip reset L[6:0] are pulled LOW and LAck, HIGH.

Data is transmitted using transition signaling on the data wires. A transition on 2 of the 7 wires conveys a symbol, which represents 4 bits of information. Finally, a transition on the acknowledge wire signals a successful transmission on receiver's side. The sender may now transmit the next one.

A symbol is encoded using transitions on L[6:0]. Table 2.1 shows each symbol and its 2-of-7 encoding. Note that a 0 represents no transition, while a 1 represents a transition on the specific wire.

The links run at around $50 Msymbols/s$, i.e. the cycle time for each symbol is around $20ns$.

2.2.3 Packet Format

Data sent on the links is formed into packets of 40 or 72 bits. The packets are conveyed as 10 data symbols (or 18 data symbols in the case of 72-bit packets) followed by an End of Packet (EoP) symbol. The format of a packet can be seen in Figure 2.3 and consists of an 8-bit header (Fig. 2.4), the packet data (32 or 64 bits) and the EoP symbol.

Symbol	$L[6]$	$L[5]$	$L[4]$	$L[3]$	$L[2]$	$L[1]$	$L[0]$
0	0	0	1	0	0	0	1
1	0	0	1	0	0	1	0
2	0	0	1	0	1	0	0
3	0	0	1	1	0	0	0
4	0	1	0	0	0	0	1
5	0	1	0	0	0	1	0
6	0	1	0	0	1	0	0
7	0	1	0	1	0	0	0
8	1	0	0	0	0	0	1
9	1	0	0	0	0	1	0
10	1	0	0	0	1	0	0
11	1	0	0	1	0	0	0
12	0	0	0	0	0	1	1
13	0	0	0	0	1	1	0
14	0	0	0	1	1	0	0
15	0	0	0	1	0	0	1
EoP	1	1	0	0	0	0	0

Table 2.1: 2-of-7 symbols encoding.

The header consists of a parity bit which ensures odd parity, a payload bit which is set for long (72-bit) packets and cleared in short (40-bit) packets, a 4-bit field dependent on the packet type which is currently not supported, thus set to “0000”, and lastly, a 2-bit field which is set to “00” for Multicast Packets.

Each packet is transmitted symbol by symbol, serially, with the first symbol being the header bits [3:0] and the last one being the EoP symbol.

Figure 2.3: SpiNNaker Link packet format.

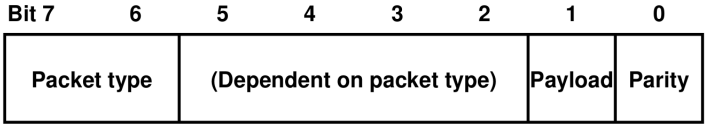


Figure 2.4: SpiNNaker Link packet header format.

2.3 Interfacing DVS to SpiNN-3 using an FPGA

After the short description of the protocols used by the DVS and the SpiNNaker platform, this section describes the development of a VLSI design on an FPGA to bridge these two protocols and implement a communication scheme between the two devices.

2.3.1 Mapping AER Events to SpiNNaker Multicast Packets

As already mentioned, events in SpiNNaker are transmitted as 40-bit packets consisting of a short 8-bit header and a 32-bit routing key. The routing key is used by the SpiNNaker routers to deliver the packets

to the correct destinations. As is the case for the AER devices, the routing key in most cases represents the origin of the event and not its destination or destinations [35].

In order to map an AER event to a SpiNNaker packet routing key, the AER device is treated as if it was a SpiNNaker chip generating events and is assigned a 16-bit virtual key. This virtual key populates bits[39:24], on the packet, and must not be shared with existing chips in the SpiNNaker system. The 16-bit event (Fig. 2.6) populates bits[23:8] on the packet injected to the SpiNNaker board. The constructed packet is shown in Figure 2.5.

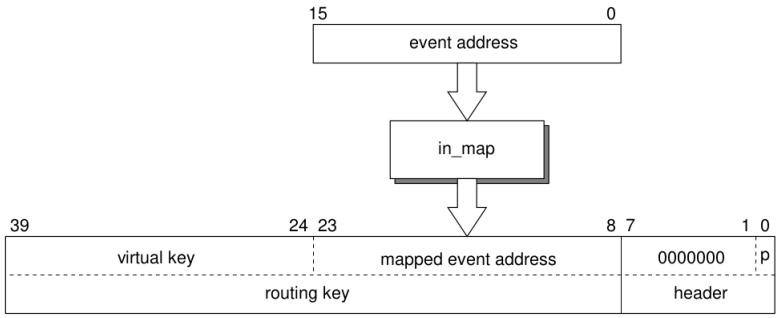


Figure 2.5: Mapping events to SpiNNaker packets.

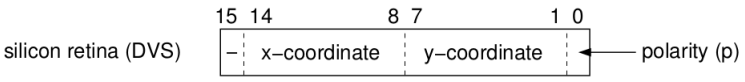


Figure 2.6: DVS event bits.

2.3.2 FPGA Design

The FPGA development board chosen for this interfacing application is the Cmod A7-35T [36] provided by Digilent (Figure 2.7). It is built around a Xilinx Artix-7 FPGA and its small size makes it suitable for mobile robotic applications. It has enough General Purpose Input Output (GPIO) pins to support the I/O functionality required by the DVS-SpiNN-3 interfacing.

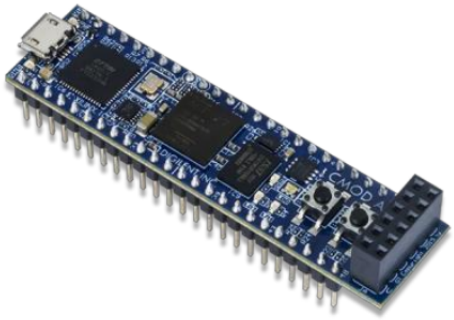


Figure 2.7: Cmod A7-35T FPGA development board.

Figure 2.8 shows the block diagram of the designed interface architecture. The Register Transfer Level (RTL) design can be seen in Appendix A. The basic modules are the *in_mapper* and the *spinn_driver*. The *in_mapper* implements the 4-phase handshake protocol to receive events from the DVS and maps them to the 40-bit packet format depicted in Figure 2.5. A *start* command can be issued by the user either from a button or by raising a specific pin externally (e.g. the main onboard PC of the robot). Doing so the *new_packet* bit is enabled. This is raised at every successful event reading and is used from

the next module, the *spinn_driver*, to initiate a new packet transmission to the SpiNN-3 board. The *spinn_driver* module breaks the packet into 10 4-bit symbols and transmits them one by one using the 2-of-7 encoding (Table 2.1) and the acknowledge signal from the SpiNN-3 (2-phase handshake). In the end, an 11_{th} packet, the EOP, is sent. During a transmission, described above, the busy bit is raised and in the end is pulled low. This is used by the *in_mapper* to prevent generating new packets in the case of communication delays imposed by the SpiNNaker. Finally, a UI module is implemented to provide the user with basic interaction with the Interface. A blue LED shows the traffic intensity from the DVS to the FPGA and a red light the traffic intensity from the FPGA to the SpiNN-3 board. Start and reset functionality is provided through buttons or software signals.

It is important to note the need of synchronizers for the incoming handshake signals in both interfaces. These asynchronous signals must be synchronised to the FPGA clock to avoid metastability. In this case, simple 2-flop synchronizers are used. Debounchers are used, too, to stabilize user input signals before feeding them into the FPGA.

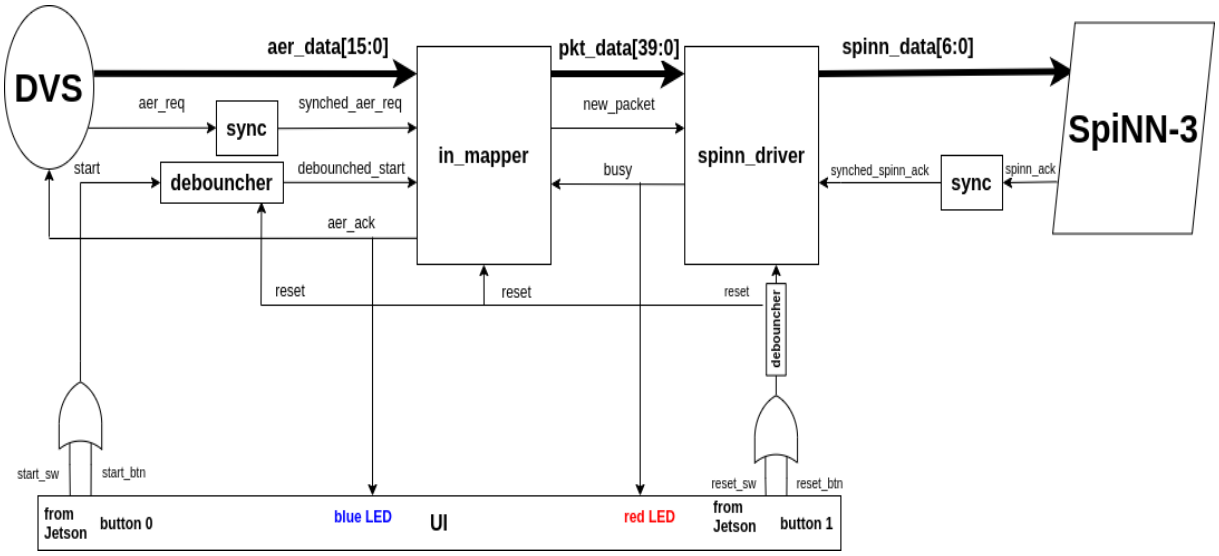
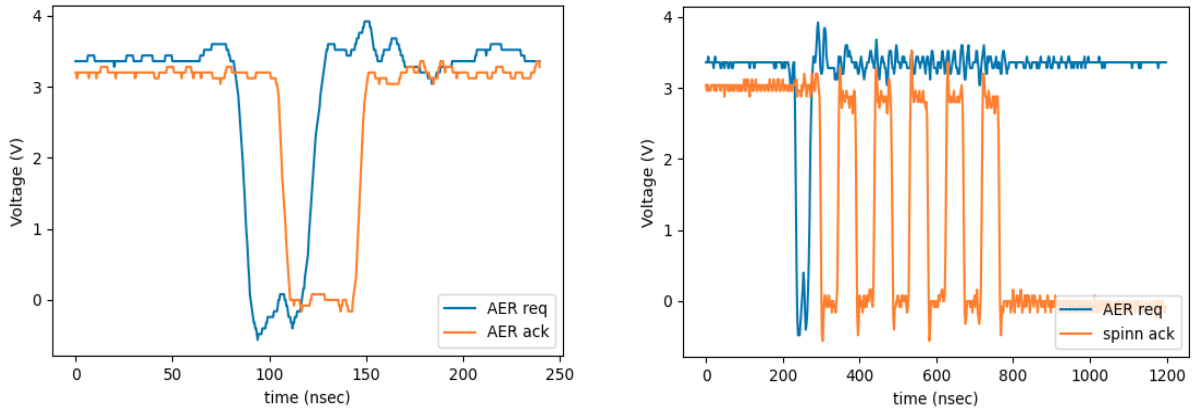


Figure 2.8: Block diagram of the DVS-SpiNN-3 interface architecture.

2.3.3 Results and System Limitations

Figure 2.9 shows the scoped request and acknowledge signals of the 4-phase handshake and 2-phase handshake protocols, of input (from DVS) and output (to SpiNN-3) accordingly. Specifically, in Figure 2.9a the DVS adds the data to be transmitted on the bus and pulls the AER request signal low. When ready, the FPGA reads the data and pulls the acknowledge signal low. Next, the DVS raises the request signal and waits till the FPGA responds to it by raising the acknowledge signal as well. A successful event reading occurred and the next event may now be transmitted. The received data are processed by the designed logic and transferred as 11 symbols to the SpiNN-3 board. SpiNN-3 toggles the acknowledge signal (Fig. 2.9b **orange**) at each successful symbol transmission. It can be seen that the 4-phase handshake lasts roughly about 50 μsec and the total DVS to SpiNN-3 latency (derived from 2.9b) is far less than 1 μsec, which is the hard system constraint.

In theory, the designed interface supports a transmission rate bigger than $1MHz$, since the total delay is less than $1\mu sec$. However, trying to communicate with SpiNN-3 at such high speed results in undefined behaviour, packets are lost and the communication is slowed down. This derives from limitations imposed by the hardware mechanism in the input of SpiNN-3, which implements the 2-phase handshake protocol and feeds the events into the cores. With a trial and error approach, it was found that the maximum data rate that the SpiNN-3 can handle without problem is about $150kHz$. Thus a specific extension was implemented on the design to apply a limit on this injection rate. The communication between the DVS and the FPGA remains unaffected. Events are freely transferred to the FPGA. But a limiting functionality is applied to the output to the SpiNN-3 board. A demonstration is shown in Figure 2.10.



(a) Input: 4-phase handshake protocol.

(b) Output: 2-phase handshake protocol.

Figure 2.9: FPGA input and output handshake protocols.

The first event (first dip in the AER request signal **(blue)**) is successfully transmitted till the SpiNN-3. The **orange** signal, the SpiNN-3 acknowledge, toggles 11 times (can not be seen in the scope's resolution). A second event occurs in less than $\frac{1}{150kHz}$, but it is not allowed to be transmitted due to the applied limit. Similarly, the third event is transmitted since there has been a sufficient time interval since the former transmitted event (the first), but the fourth is blocked, and so on.

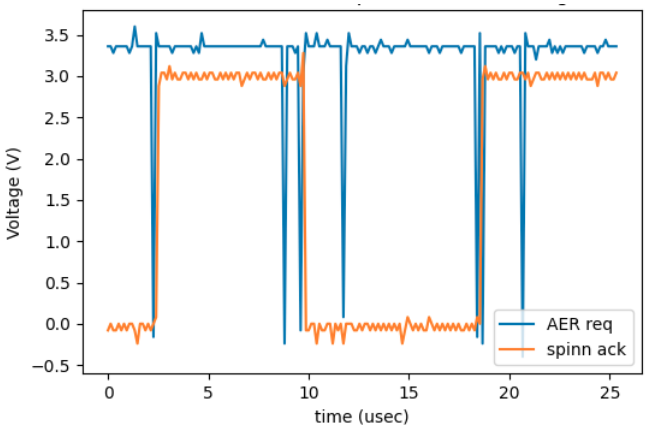


Figure 2.10: SpiNNaker data transfer rate limitation at 150 kHz.

2.4 Prototype Interface Board Design

The programmed FPGA is used to build a prototype interface board whose schematic can be seen in Figure 2.11. The DVS input is provided by an IDC20 connector and the input from Jetson AGX Xavier, by an IDC40 connector. Only two pins (*start_sw* and *rst_sw* signals) are used by the latter, but a proper full connection was applied for possible future expansion. The communication with SpiNN-3 board is established with a miniature IDC34 connector. Since the logic levels are different in the SpiNN-3 ($1.8V$) and the FPGA ($3.3V$), TXB0104 level shifters were used to apply the needed voltage level change in the SpiNNaker signals. They were chosen due to their exceptionally low latency since the application involves high frequency signals.

Using a custom 3-D printed base, the prototype and the SpiNN-3 boards are stacked together (different views are shown in Fig. 2.12). Figure 2.13 shows the complete perception system setup. It includes the DVS, the aforementioned stacked subsystem, and the onboard PC. The whole setup is designed to be mounted on the UAV, as shown in Chapter 5.

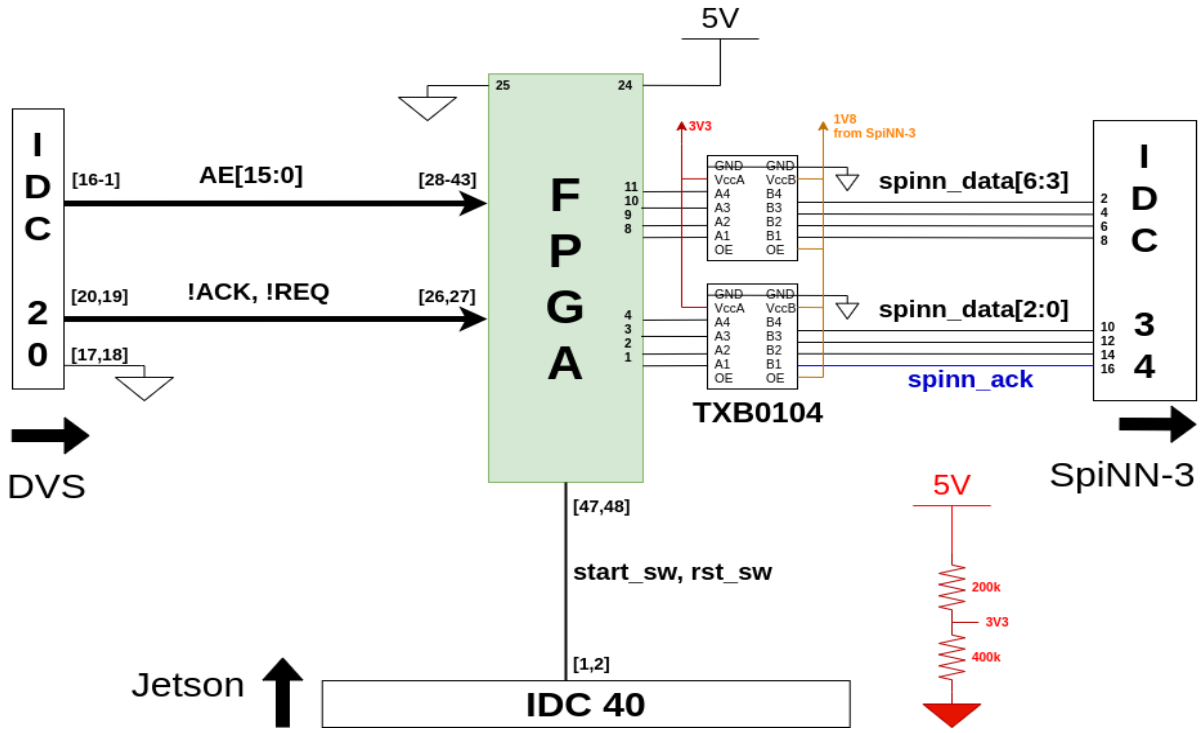


Figure 2.11: Prototype interface board schematic.

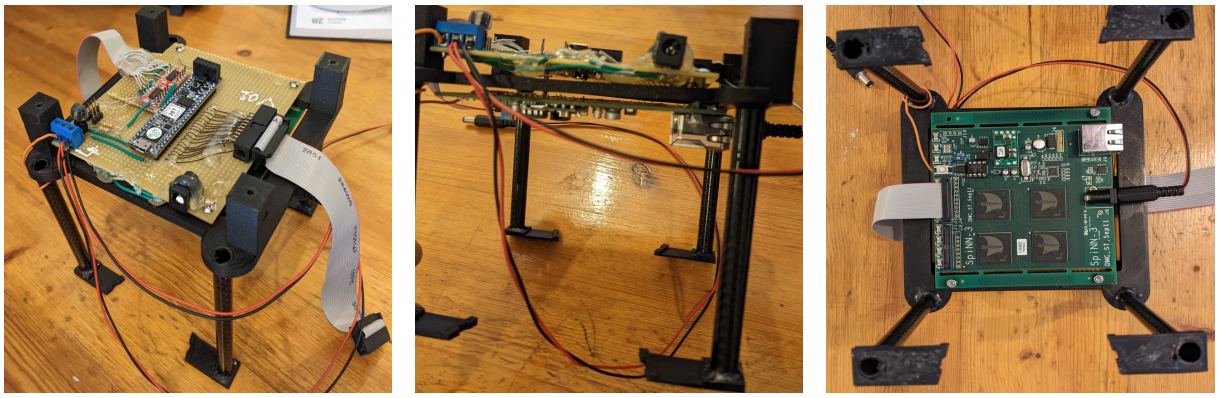


Figure 2.12: Different views of the prototype interface and the SpiNN-3 boards stacked on the mounting base.

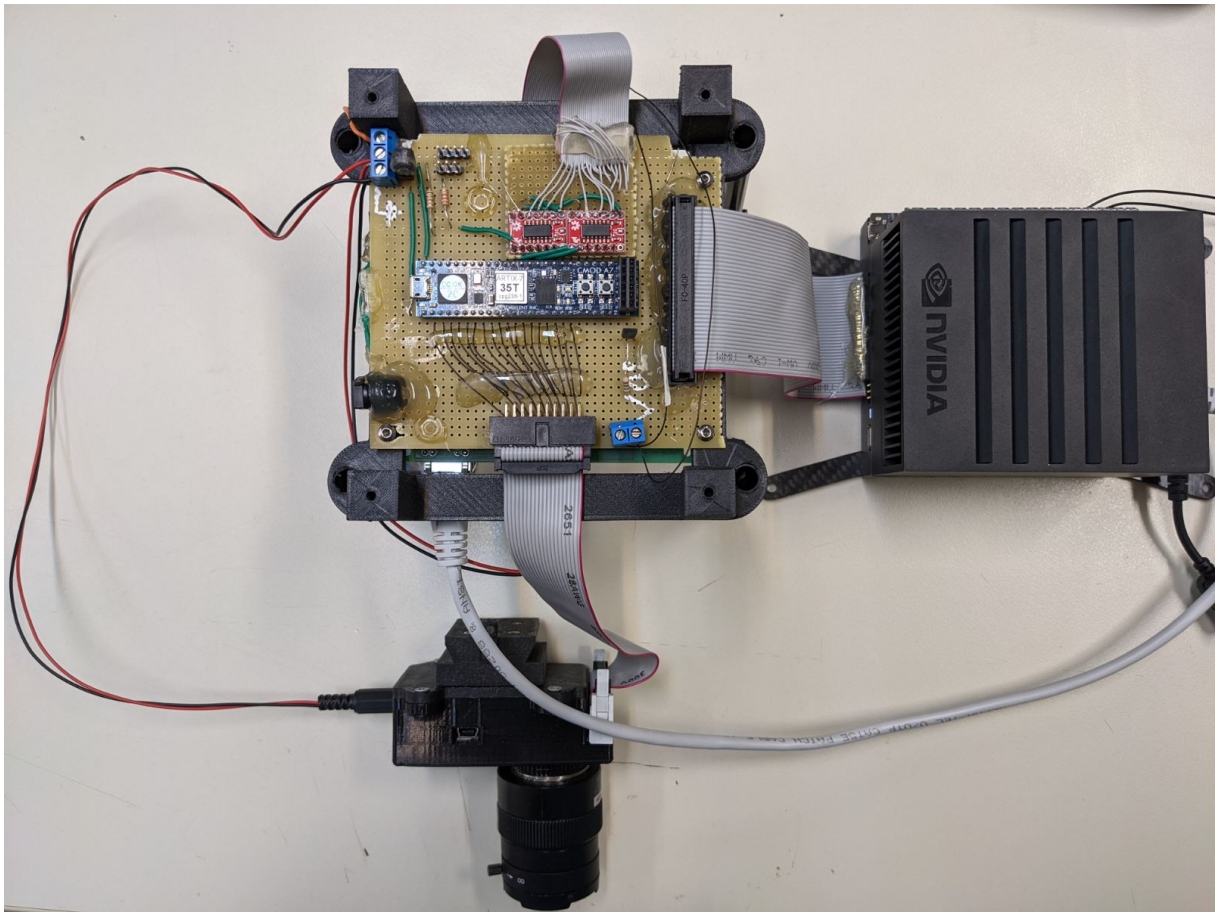


Figure 2.13: The perception system.

Chapter 3

Asynchronous Contour-Based Area Detection

The architecture of the asynchronous contour-based area detection framework is depicted in Figure 3.1.

1. The DVS-FPGA subsystem feeds events as spikes to a neuromorphic implementation of the Hough Transform (runs in the SpiNN-3 neuromorphic platform), where lines from the visual scene of both ON and OFF polarities, are detected and represented in the planar plane (r, θ).
2. The detected lines are driven into the Tracking Algorithm (runs in the onboard UAV computer - NVIDIA Jetson AGX Xavier), which eventually tracks only the desired line representing the contour-based area, i.e. pavement.
3. Moving on, the Featurer module, utilizing geometric rules extracts the desired Features (in the form of a bounding box covering the detected area - Region of Interest (ROI)).

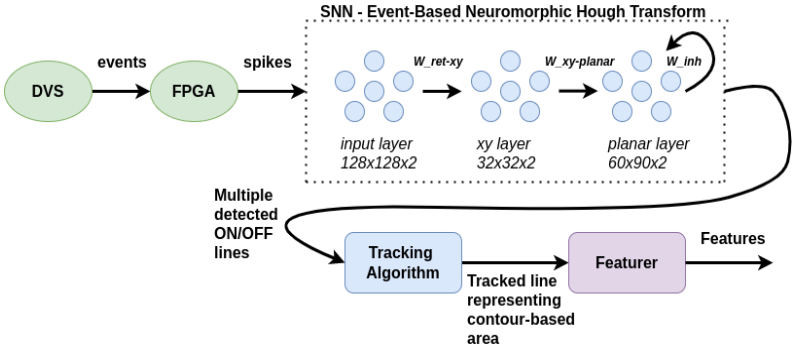


Figure 3.1: Architecture of the asynchronous contour-based area detection framework.

The detection part of the framework is based on former studies, [37] and [38], however, essential adaptations were applied to be implemented on a neuromorphic chip, as well as to align the algorithm with the real-time nature of the closed loop control system of the UAV.

3.1 Event-Based Neuromorphic Hough Transform

The main goal of the Event-Based Neuromorphic Hough Transform is to extract lines from the visual scene, from either ON or OFF events, during the flight of the UAV above a pavement. The architecture, seen in the dotted module in Figure 3.1, is used to implement the SNN executed in SpiNN-3 and is described below.

- The overall SNN consists of 3 layers, the *input* layer, the xy layer, and the *planar* layer which is also the output of the SNN.
- The *input* layer maps the events from the DVS into the neurons of the SpiNN-3 board. This is implemented implicitly by the SpiNN-3 using an appropriate hardware mechanism and not any neuron simulation hardware, thus no resource allocation occurs. It has a size of $128 \times 128 \times 2$, since the retina's dimension is 128×128 and each pixel can have a positive or negative polarity.
- The second layer, xy, implements a scale-down from the retina's $128 \times 128 \times 2$ grid to $32 \times 32 \times 2$ of simulated neurons, due to the limited number of neurons the device can simulate. To achieve the scale-down, the events coming from a 4×4 pixel grid, on the retina's side, are mapped to the same neuron of the xy layer with a weight W_{ret-xy}.
- The third layer, named *planar*, implements the basic mathematical equation of the Hough Transform, $r = xcos(\theta) + ysin(\theta)$, where $x, y \in [0, 31]$ are the coordinates of the neuron in the xy layer, and $r \in [-\frac{31}{2}\sqrt{2}, 31\sqrt{2}]$ and $\theta \in (-\pi/2, \pi/2]$ are the coordinates of the neuron in the *planar* layer. The value ranges model every possible line in the 2-D space and result from matching the dimensions of the xy layer to the polar coordinates of the Hough parameter space. Its size is $N \times M$, where N is the dimension of r and M the dimension of θ. Thus, (r, θ) values are accordingly quantized. In this implementation N and M were chosen 60 and 90 accordingly. The connection weights between the second and the third layer, $W_{xy-planar}$, derive from the aforementioned Hough Transform equation. Apparently, information from each neuron in xy layer is distributed in more than one neurons in the *planar* layer.
- The events generated by an abstract line representing a contour-based area, in this case, a pavement, are not strictly placed in a single line. In that sense, multiple lines very close to each other can be detected during the flight due to the pavement's relative motion. To address this issue, assuming that the line that best fits the area will spike first, an inhibition rectangular window of size $N_{inh} \times M_{inh}$ is applied to every neuron on the *planar* layer. Basically, each neuron is connected with all its neighbouring neurons, within the window, with an inhibitory weight, W_{inh}, to suppress the noisy detection.

The dimensions of the SNN layers, and by extension the N_{inh} and M_{inh}, were selected by taking into account the limited capabilities of the board. As mentioned before, SpiNN-3 can simulate 16k neurons, some of which are preserved by the I/O functionality of the Ethernet port. By following a trial and error procedure, about 12k neurons were finally utilized for the SNN. In an attempt to further increase the dimensions of the layers, and hence to utilize all the remaining neurons, no further improvement of the results was noticed. Therefore, the resources of the device were preserved for possible future expansion.

The tuning of the weights W_{ret-xy}, $W_{xy-planar}$ and W_{inh}, is described in section 3.4. An intuition of the proposed framework can be obtained by Figure 3.5, which provides a high level description of the information flow through each stage, thus it will be referenced in the next sections as well. Three non-trivial situations were considered in this example. The first row captures the pavement and a light pole shadow, the second the pavement and a road sign shadow, and the third the pavement and a tree shadow. Subfigure 3.5a depicts the accumulated events over a small $\Delta t = 50msec$ (accumulation applied only for visualization). Subfigure 3.5b shows the output of the SNN described in this section. Each spiking neuron models a unique line of specific polarity on the 2-D plane. It is clear that the network detects the pavement every time, however, it detects other lines too, without making any distinction.

3.2 Tracking Algorithm

The output of the Event-Based Neuromorphic Hough Transform module is directed into the input of the Tracking module (Fig. 3.1). The challenge, initially, is to detect the pavement among all the ON and OFF detected lines and afterwards to track it by updating its values according to the input spikes. It is crucial that this is applied while also optimizing the algorithm in terms of speed. To apply a solution to this idea, the algorithm developed in [37] is chosen and has been extended accordingly to adapt to the real-time nature of the application. The extended version, developed in this thesis, is described below and is formulated mathematically in Algorithm 1.

Clusters $C_p = (T, R, \Theta, T_{exp}, N)$ are used to represent the various tracked lines of polarity $p \in [ON, OFF]$. Whenever a spike $s = (t, r, \theta)$ occurs, the algorithm tries to associate the new line with an existing cluster by checking the Euclidean distance between (r, θ) and (R, Θ) of all clusters, given that the distance is lower than a threshold value $D_{threshold}$.

Parameter $\alpha \in (0, 1]$ affects the amount of impact each spike has on the values of the respective cluster (line 9 of Algorithm 1), T_{exp} parameter is used to identify clusters that have not been updated for T_{hidden} time and are thus considered expired and must be discarded. In contrast, the N parameter is incremented each time a spike is associated with a specific cluster till it reaches a K value. Then this cluster is considered valid and all others of the same polarity are discarded. Successful pavement detection is defined by the existence of both an ON and an OFF cluster. Finally, the pavement's (r, θ) values are calculated as the average clusters' values (lines 24 and 25 of Algorithm 1).

The generated event stream is mostly populated by spikes generated by the relative motion of the pavement, thus by choosing relatively high N and small T_{hidden} values, the pavement is correctly detected, without false positives, each time the UAV hovers over it. As soon as the pavement is correctly detected, the algorithm avoids generating new clusters as this would only increase the complexity of the algorithm. However, it continues associating new spikes with the two clusters that define the pavement, thus continuously updating its position.

Figures 3.5b and 3.5c qualitatively depict the functionality of the described algorithm. Especially, in the case of the pavement plus the road sign shadow it is obvious that lines irrelevant to the pavement are rejected while the ones associated with it update its position.

Algorithm 1 Tracking algorithm

1: **for** every spike $s = (t, r, \theta)$ **do**
2: find the polarity $p = ON$ or OFF of the spiked neuron
3: delete all clusters $C_p = (T, R, \Theta, T_{exp}, N)$ with $T_{exp} < t$
4: find the minimum Euclidean distance D between (r, θ) and all $C_p(R, \Theta)$
5: find the winner cluster, accordingly
6: **if** $D \leq D_{threshold}$ **then**
7: spike s belongs to the winner cluster
8: **for** the winner cluster **do**
9: $(R, \Theta) = (1 - a)(R, \Theta) + a(r, \theta)$
10: $T = t$
11: **if** $pavement_p$ not detected **then**
12: $N = N + 1$
13: $T_{exp} = t + T_{hidden}$
14: **if** N = K **then**
15: $pavement_p$ detected
16: set T_{exp} to a big value
17: **end if**
18: **end if**
19: **end for**
20: **else if** $pavement_p$ not detected **then**
21: generate a new cluster $C = (t, r, \theta, t + T_{hidden}, 0)$
22: **end if**
23: **if** $pavement_p$ is detected for both ON and OFF polarities **then**
24: $R_{pavement} = (C_{ON}(R) + C_{OFF}(R))/2$
25: $\Theta_{pavement} = (C_{ON}(\Theta) + C_{OFF}(\Theta))/2$
26: **end if**
27: **end for**

3.3 Features Extraction

As soon as the pavement is detected and successfully tracked by the SNN and the tracking algorithm, the features needed for the IBVS controller, discussed in Chapter 4, should be extracted. An appropriate module, the Featurer, is designed for that purpose. Its functionality is described below and formulated mathematically in Algorithm 2.

Initially, r and θ values get de-quantized. A scale up from 32×32 grid to 128×128 grid is applied for the value r, to return from the reduced size to the original one. Using the estimated (r, θ) values we define two lines, in the uv Cartesian space, situated on both sides of the detected line at a distance $\frac{C}{2}$ from it (Fig. 3.2a). The idea is that these two lines form a trapezoid that roughly bounds the contour-based area. In its most probable state, this will be a parallelogram (Fig. 3.2b) of base C. To find the 4 points forming the bounding box the cross-sections of these two lines with the axes $u = 0, u = 127, v = 0, v = 127$ are calculated, while rejecting the points with coordinates outside the set [0,127]. The resulting 4 points form the set $P = \{p_0, p_1, p_2, p_3\}$ which is the unordered feature vector.

To order it, the following logic is applied. We assume that the desired ordered feature vector is $D = \{d_0, d_1, d_2, d_3\}$. Afterwards, the sum of the Euclidean distances of every point on D set to the according point on R set is calculated. This is done for every possible mapping of the P set in D. The mappings that result in the minimum distances summation are the candidates for the ordered set. Finally, by finding the area that each ordered candidate set covers and keeping the biggest one, the correct ordered feature vector is extracted.

This method works correctly in the general case too, where the detected line can be anywhere in the 2-D plane. In that case, the feature vector forms a trapezoid, of height C.

Figure 3.5d qualitatively shows the features extracted by the detected lines representing the contour-based areas (pavement).

Algorithm 2 Featurer module algorithm

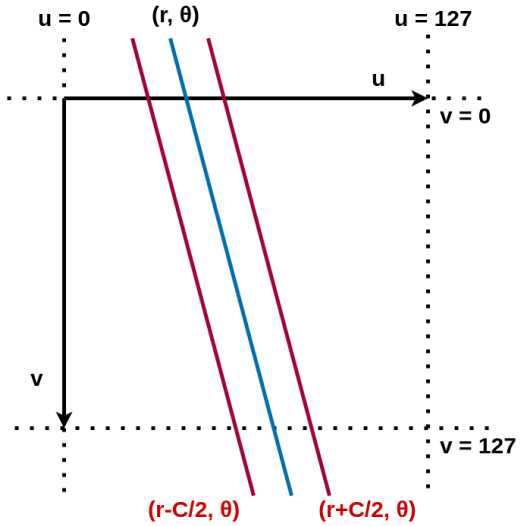
1: $(r, \theta) = (R_{pavement}, \Theta_{pavement})$
2: r and θ get de-quantized
3: r is scaled up to 128×128 grid
4: define C, the height of the trapezoid ROI
5: calculate $(r - \frac{C}{2}, \theta)$ and $(r + \frac{C}{2}, \theta)$ lines in the uv Cartesian space
6: find the points in the cross-sections of the lines $(r - \frac{C}{2}, \theta)$ and $(r + \frac{C}{2}, \theta)$, and the axes $u = 0, u = 127, v = 0, v = 127$
7: reject points whose u or v coordinate is outside the set [0,127]. The remaining set of points, say, $P = \{p_0, p_1, p_2, p_3\}$ are the 4 points of the unordered feature vector
8: assume $D = \{d_0, d_1, d_2, d_3\}$ is the ordered set of points that form the desired feature vector
9: assume $R = \{r_0, r_1, r_2, r_3\} = \{(64 + \frac{C}{2}, 127), (64 - \frac{C}{2}, 127), (64 - \frac{C}{2}, 0), (64 + \frac{C}{2}, 0)\}$ is the set of points forming the reference feature vector
10: find the minimum $C_n = \sum_{i=1}^{4} Dist(d_i, r_i)$ $\forall$ mapping of set P in set D, where $Dist$ is the Euclidean distance of the two points and $n \in [1, 4!]$
11: for the sets $D_j = \{(d_{0_x}, d_{0_y}), (d_{1_x}, d_{1_y}), (d_{2_x}, d_{2_y}), (d_{3_x}, d_{3_y})\}_j$ that result in the minimum C find the area covered by each one using the formula

$$A_j = \{\frac{1}{2}(d_{0_x}d_{1_y} - d_{1_x}d_{0_y} + d_{1_x}d_{2_y} - d_{2_x}d_{1_y} + d_{2_x}d_{3_y} - d_{3_x}d_{2_y} + d_{3_x}d_{0_y} - d_{0_x}d_{3_y})\}_j$$

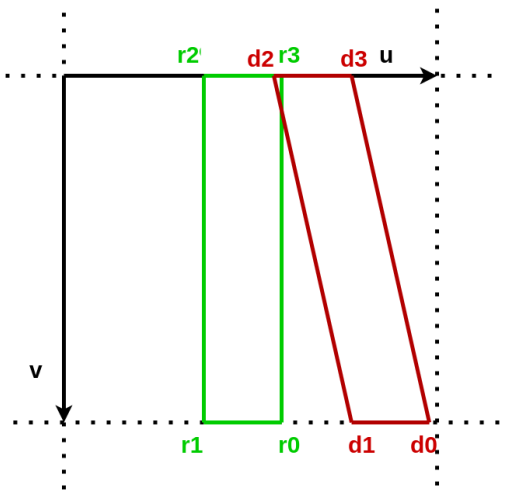
12: the set that covers the maximum area A is the set of points that form the desired feature vector

3.4 Tuning and Performance

The overall architecture of both the SNN and the tracking algorithm has been established in Sections 3.1 and 3.2, respectively. The last step is to train the hybrid detection system using recorded data. All in all,

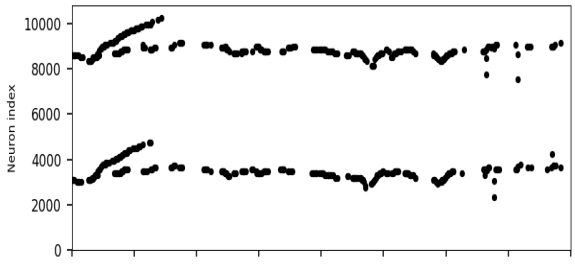


(a) Creation of the unordered feature vector.

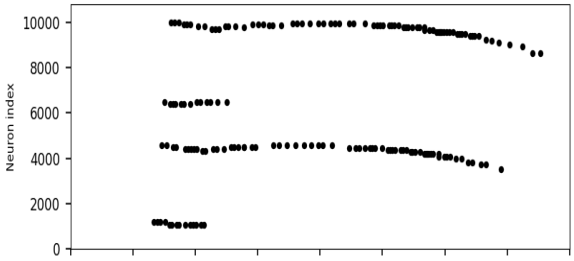


(b) Desired and reference feature vectors.

Figure 3.2: Geometrical features extraction.



(a) Pavement following.



(b) Yaw rotation.

Figure 3.3: SNN output on two UAV teleoperation scenarios.

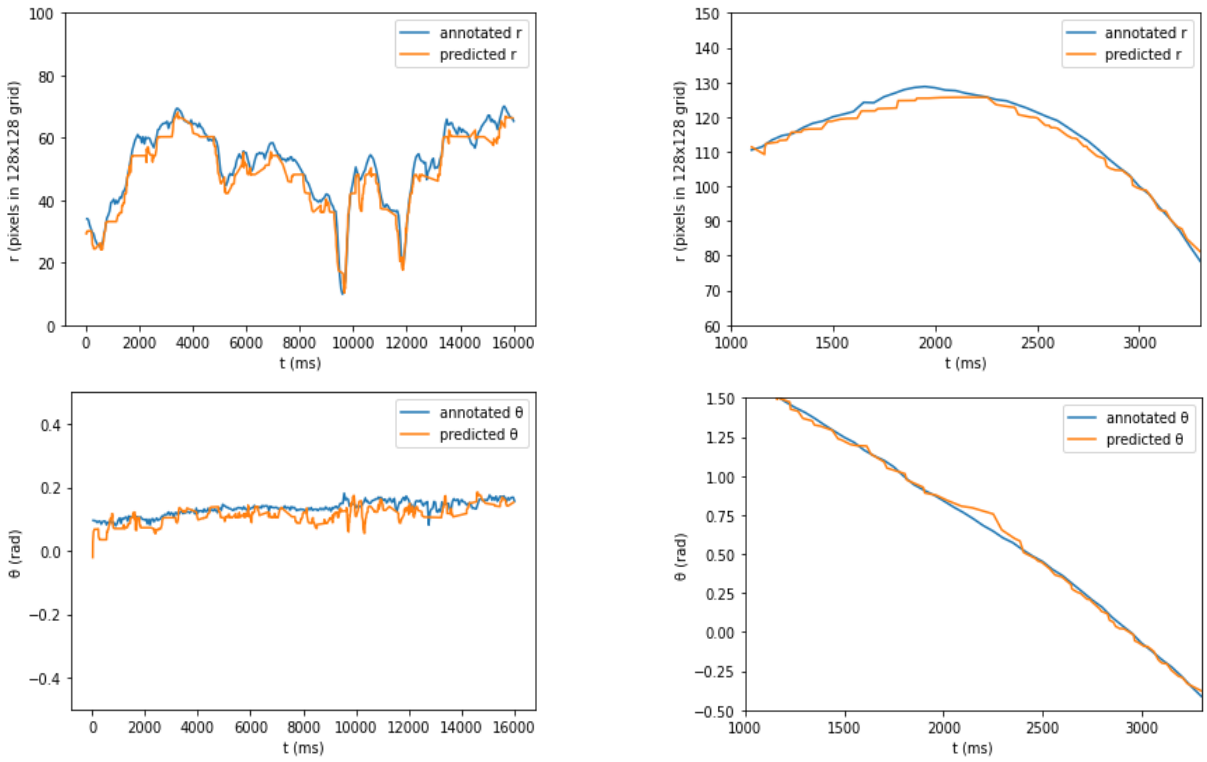
the perception algorithm has 8 trainable parameters, W_{ret-xy}, $W_{xy-planar}$, W_{inh}, $D_{threshold}$, α, T_{hidden}, N and K. Thus, the problem turns into a tuning task and can be easily approached with a simple brute force method. In this direction many value combinations for the above parameters were examined, within some predefined rational ranges, using the a custom metric.

Using a small DVS recording of a flight, the development of this metric is achieved as follows. At first, events are accumulated over a few milliseconds (small enough to ensure almost no movement of the pavement for that time frame). The pavement's position is annotated in each frame manually and is finally used as a metric for obtaining the near-optimum values for the 8 tunable parameters of the algorithm.

With the algorithm tuned, the perception system is tested under 2 different scenarios. The first one is a pavement surveillance motion and the second is a rotation on the yaw angle combined with small lateral displacement. In both cases, the UAV is teleoperated. Figure 3.3 shows the output of the proposed SNN for both motions. The figures show the spikes occurring throughout the task, while Y-axis denotes the index of the spiked neuron, as this is modeled in SpiNN-3 device by the following expression.

$$neuron\_id\_planar = n + 60m + 5400p,$$

where n and m are the coordinates of the spiking neuron in the 60×90 grid and $p \in \{0, 1\}$ is the polarity (ON or OFF). The r and θ information, deriving from the Figures, is coupled so they do not provide a lot of meaningful observations. However, it is worth noticing that the designed SNN achieves high levels of noise reduction and continuous pavement detection. Shadows and other lines of the environment are detected, too, as is clearly extracted by these Figures.



(a) Pavement following.

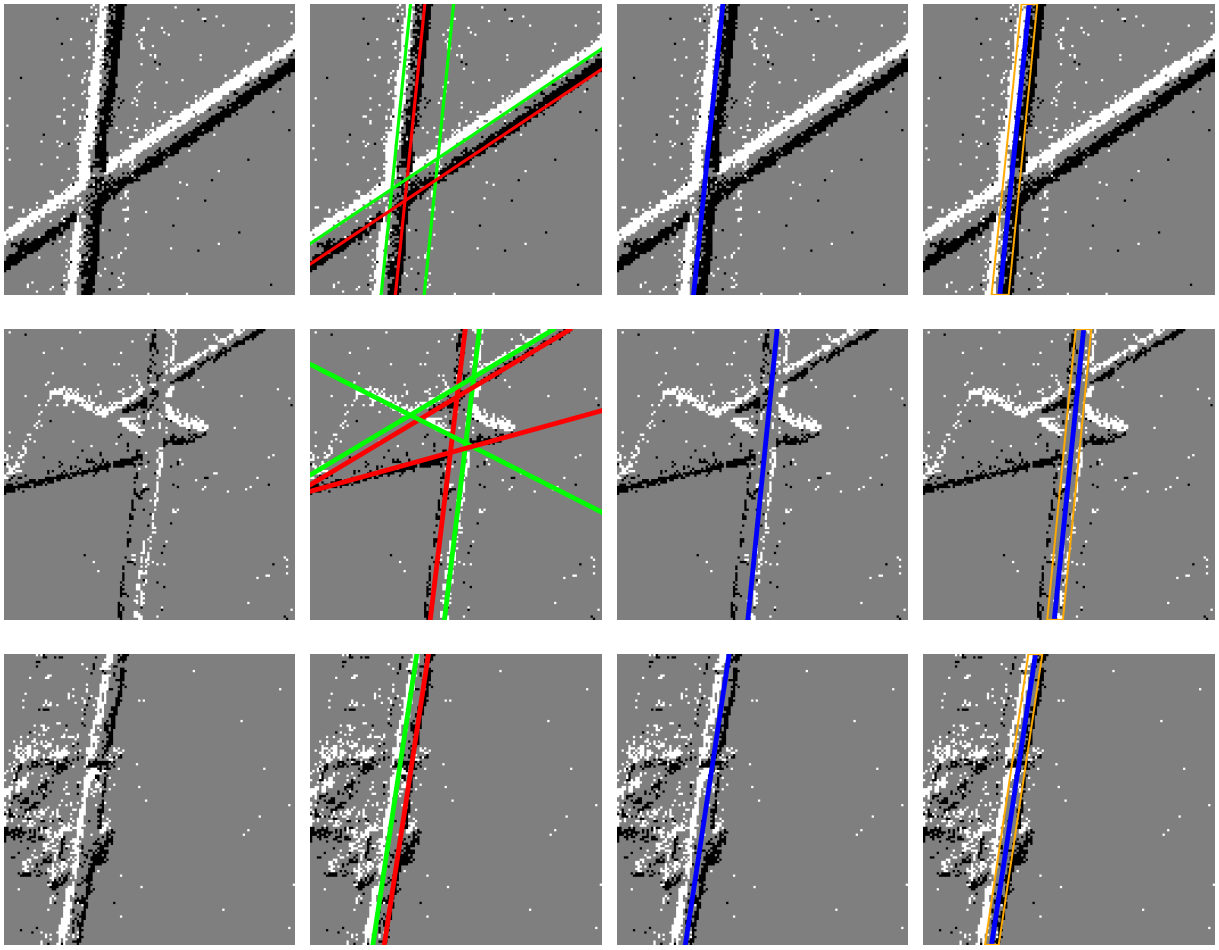
(b) Yaw rotation.

Figure 3.4: Detection framework performance on two UAV teleoperation scenarios. The predicted values **(orange)** follow successfully the annotated **(blue)**. In the first case θ is almost zero since the UAV is controlled by a human to follow the pavement, while r performs small fluctuations produced by the sway motion of the UAV. In the second case, due to the yaw rotation, θ is linearly decreasing, while r is changing, too, under the effect of the combined small lateral motion. ***Important: r seems to fluctuate a lot due to the small FoV of the DVS (approximately*** $1.5m \times 1.5m$ ***at height*** $5m$***).***

The latency of the perception system can be calculated as follows. The interface design imposes a delay, lower than $1\mu sec$ and can be safely ignored. The SNN running on the SpiNN-3 performs calculations at $1msec$ timesteps. In each timestep, information is conveyed from one layer to the next. Moreover, performing measurements on the implementation of the tracking algorithm is found that the calculations are completed in less than $1msec$. Considering the above, and the number of layers in the SNN, the overall perception system latency is calculated at $5msec$. However, the update rate, limited only by the tracking algorithm speed, is asynchronous and can reach up to $1kHz$ depending on the SNN output spikes frequency.

The successful performance of the detection framework is qualitatively depicted in Figure 3.5c, where the blue line indicates the detected line representing the pavement. Figure 3.4 quantitatively captures the capabilities of the proposed detection framework by comparing the (r, θ) sets of values, for the detected pavement, between the annotated and the predicted case. Indicatively, for both r and θ an error below **4%** is noticed. The prediction follows almost perfectly the annotated line even in rapid movements of the UAV (Fig. 3.4a, pavement following), with negligible delay. The results are similar in the second case of the yaw rotation 3.4b.

It is worth mentioning here, that apart from its performance, due to its simplicity, the proposed framework is very versatile and can be tuned for various environments with just a small amount of recorded data, as proven in this Section.



(a) (b) (c) (d)

Figure 3.5: DVS events from the visual scene **(a)** are given as input to the event-based Hough Transform, resulting in multiple detected lines **(b)**. The green lines are extracted from ON events and the red lines from OFF events. The output **(c)** of the tracking algorithm results in the features **(d)** in the form of a bounding box. The first row captures the pavement and a light pole shadow, the second the pavement and a road sign shadow, and finally the third the pavement and a tree shadow.

Chapter 4

Control Strategy - IBVS

4.1 Control Law

Visual Servo Control refers to a control approach that is based on computer vision, image processing and control theory, and utilizes computer vision data in the closed loop to control the movement of a robot. There are various configurations for the placement of the camera sensor relative to the position of the robot. The one used in the setup of the UAV in this thesis is the so-called eye-in-hand. In this case, the camera is mounted on the end-effector of the robotic manipulator and records the position of the target relative to the robot. The two basic visual servo controllers are the Image-Based Visual Servo (IBVS) and the Position-Based Visual Servo (PBVS) controller ([39], [40]). The formulation that follows corresponds to the eye-in-hand case for an IBVS controller since this approach is the one followed by this thesis.

The aim of all vision-based control schemes is to minimize an error $\mathbf{e}(t)$, which is typically defined by

$$\mathbf{e}(t) = \mathbf{s}(\mathbf{m}(t), \mathbf{a}) - \mathbf{s}^*. \tag{4.1}$$

The vector $\mathbf{m}(t)$ is the set of image measurements of the interest points (in pixels) that form the bounding box surrounding the contour-based area (Chapter 3 Section 2). These image measurements are used to compute a vector of k visual features, $\mathbf{s}(\mathbf{m}(\mathrm{t}), \mathbf{a})$, in which $\mathbf{a}$ is a set of parameters that represent potential additional knowledge about the system and, here, is nothing but the camera's intrinsic parameters. The vector $\mathbf{s}^*$ contains the desired values of the features. This study involves a motionless target (bounding box located in the center of the image) thus $\mathbf{s}^*$ is constant and changes in $\mathbf{s}$ depend only on the motion of the camera (hence the motion of the UAV).

Since $\mathbf{s}$ is defined, the control scheme is designed as a velocity controller. To do this, we require the relationship between the time variation of $\mathbf{s}$ and the camera velocity. Let the latter be denoted by $\mathbf{v}_c = (\mathbf{t}_c, \boldsymbol{\omega}_c)$, with $\mathbf{t}_c$ the instantaneous linear velocity of the origin of the camera frame and $\boldsymbol{\omega}_c$ the instantaneous angular velocity of the camera frame. The relationship between $\dot{\mathbf{s}}$ and $\mathbf{v}_c$ is given by

$$\dot{\mathbf{s}} = \mathbf{L}_\mathbf{s}\mathbf{v}_c, \tag{4.2}$$

in which $\mathbf{L}_\mathbf{s} \in \mathbb{R}^{k\times 6}$ is the interaction matrix related to $\mathbf{s}$.

Combining Equations 4.1 and 4.2, the relationship between the camera velocity and the time variation of the error is derived.

$$\dot{\mathbf{e}} = \mathbf{L}_\mathbf{e}\mathbf{v}_c, \tag{4.3}$$

where $\mathbf{L}_\mathbf{e} = \mathbf{L}_\mathbf{s}$. Considering $\mathbf{v}_c$ as the input to the robot controller and assuming a desired exponential decoupled decrease of the error (i.e. $\dot{\mathbf{e}} = -\lambda\mathbf{e}$), we obtain using 4.3

$$\mathbf{v}_c = -\lambda\mathbf{L}_\mathbf{e}^+\mathbf{e}, \tag{4.4}$$

where $\mathbf{L_e^+} \in \mathbb{R}^{6\times k}$ is chosen as the Moore-Penrose pseudo-inverse of $\mathbf{L_e}$, that is $\mathbf{L_e^+} = (\mathbf{L_e^T L_e})^{-1}\mathbf{L_e^T}$ when $\mathbf{L_e}$ is of full rank 6. In the case of $k = 6$, if $\det\mathbf{L_e} \neq 0$ it is possible to invert $\mathbf{L_e}$, thus the control law becomes $\mathbf{v}_c = -\lambda\mathbf{L_e^{-1}e}$.

In real visual servo systems, however, it is impossible to know perfectly in practice either $\mathbf{L}_e$ or $\mathbf{L}_e^+$. So an approximation or an estimation must be realized. We denote both the approximation of the pseudo-inverse of the interaction matrix and the pseudo-inverse of the approximation of the interaction matrix by the symbol $\widehat{\mathbf{L}_e^+}$, thus the control law finally becomes

$$\mathbf{v}_c = -\lambda\widehat{\mathbf{L}_e^+}\mathbf{e}. \tag{4.5}$$

4.2 Interaction Matrix

Adopting a central projection perspective model (described in [41]) for the camera, a 3-D point with coordinates $\mathbf{X} = (X, Y, Z)$ in the camera frame projects in the image plane as a 2-D point with coordinates $\mathbf{x} = (x, y)$. It is derived

$$\begin{cases} x = \frac{X}{Z} = \frac{u-c_u}{f\alpha} \\ y = \frac{Y}{Z} = \frac{v-c_v}{f} \end{cases}, \tag{4.6}$$

where the image measurement $\mathbf{m} = (u, v)$ is expressed in pixels and $\mathbf{a} = (c_u, c_v, f, \alpha)$ is the set of camera intrinsic parameters: c_u and c_v are the coordinates of the principal point, f is the focal length, and α is the ratio of the pixel dimensions.

Taking the time derivative of equations 4.6 we obtain

$$\begin{cases} \dot{x} = \frac{\dot{X}}{Z} - \frac{X\dot{Z}}{Z^2} = \frac{\dot{X}-x\dot{Z}}{Z} \\ \dot{y} = \frac{\dot{Y}}{Z} - \frac{Y\dot{Z}}{Z^2} = \frac{\dot{Y}-y\dot{Z}}{Z} \end{cases}. \tag{4.7}$$

We can relate the velocity of the 3-D point to camera spatial velocity using the following equation

$$\dot{\mathbf{X}} = -\mathbf{t}_c - \boldsymbol{\omega}_c \times \mathbf{X} \Leftrightarrow \begin{cases} \dot{X} = -t_x - \omega_y Z + \omega_z Y \\ \dot{Y} = -t_y - \omega_z X + \omega_x Z \\ \dot{Z} = -t_z - \omega_x Y + \omega_y X \end{cases}. \tag{4.8}$$

Combining 4.7 with 4.8 it is obtained

$$\begin{cases} \dot{x} = -\frac{t_x}{Z} + \frac{xt_z}{Z} + xy\omega_x - (1+x^2)\omega_y + y\omega_z \\ \dot{y} = -\frac{t_y}{Z} + \frac{yt_z}{Z} + (1+y^2)\omega_x - xy\omega_y - x\omega_z \end{cases} \tag{4.9}$$

which can be expressed in the following compact form

$$\dot{\mathbf{x}} = \mathbf{L_x}\mathbf{v}_c, \tag{4.10}$$

where the interaction matrix $\mathbf{L_x}$ related to $\mathbf{x}$ is

$$\mathbf{L_x} = \begin{bmatrix} -\frac{1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & -\frac{1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{bmatrix}. \tag{4.11}$$

The value Z is the depth of the point relative to the camera frame. Moreover, x and y are computed given the intrinsic parameters of the camera. Therefore the control scheme that uses this form of the interaction matrix must estimate or approximate the aforementioned values to finally make use of the estimation or the approximation of the interaction matrix, $\widehat{\mathbf{L_x}}$.

To control the 6 DoF, at least three points are necessary (i.e $k \geq 6$). In this case, 4 points are used (the 4 corners of the ROI). The feature vector becomes $\mathbf{s} = \mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$ and the interaction matrix is obtained by stacking the 4 interaction matrices related to each feature

$$\mathbf{L_x} = \begin{bmatrix} \mathbf{L}_{\mathbf{x}_1} \\ \mathbf{L}_{\mathbf{x}_2} \\ \mathbf{L}_{\mathbf{x}_3} \\ \mathbf{L}_{\mathbf{x}_4} \end{bmatrix}. \tag{4.12}$$

The Z value is the altitude of the robot and is available through sensor measurements at each iteration of the control scheme. Camera intrinsic parameters are available, too, hence we can assume that $\mathbf{L_e} = \mathbf{L_x}$ is known and subsequently $\widehat{\mathbf{L_e^+}} = \mathbf{L_e^+}$. Using this estimation, the control law (Equation 4.5) can be used to control the 6 DoF robot motion. Stability analysis of the IBVS, shown in [39], proves that the method can achieve local asymptotic stability. Determining the size of the neighborhood where stability and convergence are ensured is still an open issue.

It is worth mentioning that in the general case the frame of the UAV does not coincide with the camera frame. For this reason, camera velocity commands $\mathbf{v}_c$ must be transformed to the UAV coordinate frame using the appropriate translation and rotation matrices before being given to the robot controller.

4.3 Low-Level Multirotor Control

The IBVS controller is the higher level control module of the UAV. A set of cascaded PID controllers is responsible for handling the low-level control of the vehicle via the following architecture [42]:

- An inner loop executing attitude control while using as input references roll, pitch, yaw, and throttle values. The output of this loop is used as voltage input to the motor controller of the propellers.
- An outer loop executing translational motion control while using as input references the desired position or velocity values.

This architecture enables an IBVS approach since the outer loop can receive velocity inputs w.r.t the UAV coordinate frame. These velocities are generated by the IBVS algorithm after they have been transformed from the image to the fixed-body coordinate frame of the UAV. Furthermore, this low-level control architecture handles successfully the under-actuation problem of the multirotor.

Chapter 5

Experimental Results

5.1 Experimental Setup

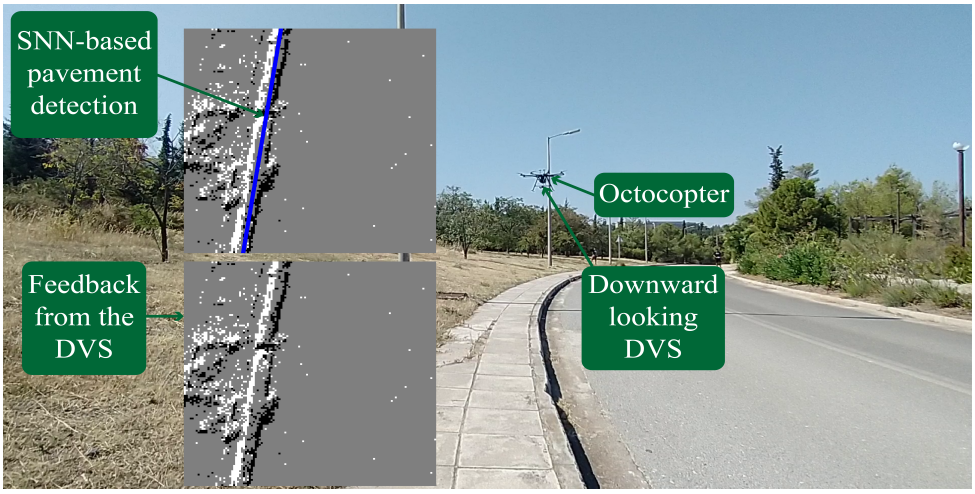


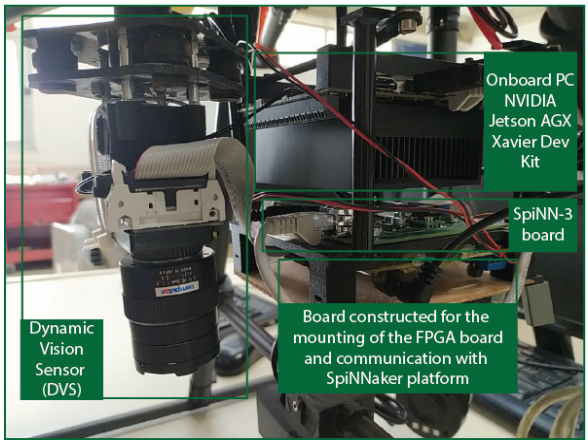
Figure 5.1: Octorotor UAV executing pavement detection and tracking.

The efficacy of the proposed framework is demonstrated through a series of outdoor experiments in various environment sets deploying a custom-made octorotor equipped with the developed asynchronous perception system, and a Pixhawk Cube Orange featuring ArduPilot firmware (Fig. 5.2b). The hardware of the perception system, depicted in Figure 5.2a, consists of the DVS, the ptototype interface board, the SpiNN-3 platform and an onboard computer (NVIDIA Jetson AGX Xavier).

In general, the SpiNN-3 board is used to implement the core of the detection algorithm, using the Hough Transform. The onboard computer is used to run the tracking algorithm on the outcome of the SpiNN-3 board and simultaneously the IBVS control strategy using ROS. The MAVROS [43] communication protocol is used to send the velocity commands produced by the IBVS control scheme to the low level controller of the octocopter. The proposed control strategy architecture is depicted in Figure 5.3.

5.2 Results

To test and demonstrate the performance of the proposed control strategy, two sets of experiments were conducted on different pavement settings. The vehicle was flying at a relatively low altitude (approximately $5m$), utilizing the IBVS control scheme. Using the output of the detection algorithm the extracted features of the contour-based area determine the Moore-Penrose pseudo-inverse of the interaction matrix



(a) Perception system mounted on the octorotor UAV.

(b) Octorotor UAV in testing area.

Figure 5.2: Custom octorotor UAV setup.

$\widehat{\mathbf{L}_\mathbf{e}^+}$ and the error $\mathbf{e}$, in all circumstances. For all features, $\mathbf{s}_i$, $i = 1, 2, 3, 4$, the depth measurements Z_i are considered equal and collected by the octocopter's altimeter sensor. The purpose of the controller is to minimize the error of the feature vector. Consequently, to achieve this goal, the edges of the bounding box must approach a set of desired coordinates on the image plane, which were properly set around the center of the image.

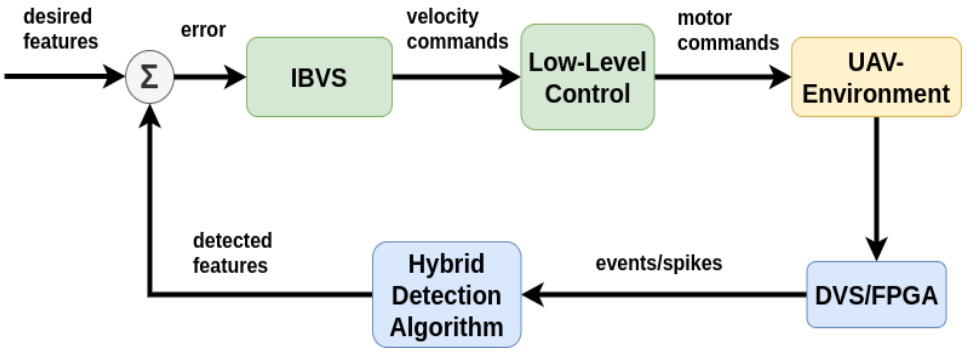


Figure 5.3: Closed loop system architecture.

Experiment 1

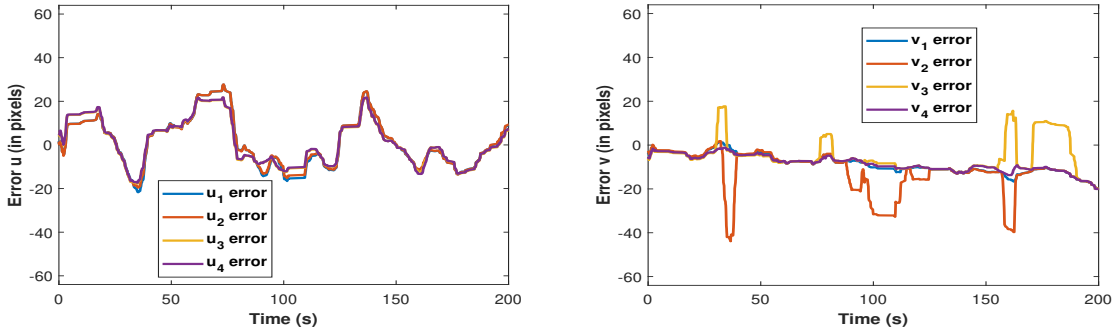
At first, the location shown in Figure 5.4a is selected for demonstration. The target is a straight pavement with various challenges (trees, shadows, e.t.c.). Figure 5.5 shows the error in pixels of the u and v coordinates of the feature vector. It can be seen that the UAV completed its task successfully, following the pavement while constantly keeping the detected contour inside the FoV of the DVS. Furthermore, the error is kept sufficiently low resulting in a smooth flight with minor oscillations. Note that the error observed in the Figures translates to minimal oscillations on the UAV, since the FoV is very small (approximately $1.5m \times 1.5m$ at a height $5m$).

Experiment 2

A second demonstration scenario (Fig. 5.4b) in a different pavement location is conducted to prove the robustness of the control strategy. Figure 5.6 proves the repeated success of the method in a more demanding location than the corresponding first one, due to target curvature.

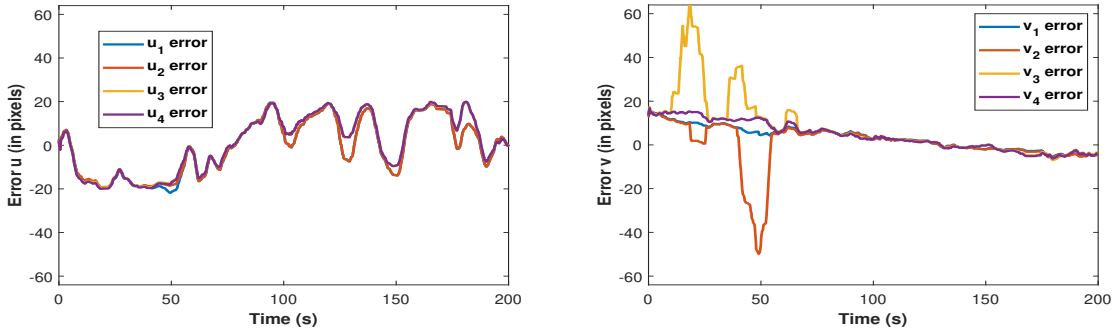
(a) First experiment location. (b) Second experiment location.

Figure 5.4: Experiment locations.



(a) (b)

Figure 5.5: Experiment 1: Error features along u-axis (5.5a), v-axis (5.5b) during the first experimental scenario conducted in a pavement setting.



(a) (b)

Figure 5.6: Experiment 2: Error features along u-axis (5.6a), v-axis (5.6b) during the second experimental scenario conducted in a pavement setting.

Chapter 6

Conclusion

6.1 Assessment

In this work, an end-to-end control strategy for the efficient tracking of contour-based areas using a UAV is presented. The proposed perception framework, senses, detects, and tracks contour-based areas, such as pavements, with various characteristics, without the use of a central clock. A bio-inspired DVS camera acquires events of the scene. An SNN running on a SpiNN-3 board is utilized to process the DVS event stream and the outcome is used by a tracking algorithm to further de-noise it and track the line that represents the contour-based area. The efficiency of the resulting perception system is tested under different scenarios and achieved very good performance with detection error under 4%, even under rapid movements of the UAV, with negligible delay. Specifically the total perception system latency is $5msec$ with an asynchronous update rate up to $1kHz$. At the same time, the power consumption is far less than a classic CNN-based detection scheme, due to the combination of the event-based nature of the sensor with the neuromorphic hardware. The reduced power consumption applies in the training/tuning phase of the algorithm, too, since it can be implemented using a small set of recorded data.

A geometric-based algorithm is applied to extract the respective features that form the error used to develop an IBVS scheme. The developed IBVS controller exploits the high update rate and the small delay of the detection framework. The efficiency of the proposed control architecture is demonstrated successfully by a series of real-time experiments conducted in various environments using an octorotor UAV equipped with the appropriate sensor and neuromorphic hardware suite.

6.2 Future Work

Even though the detection framework was developed for pavement tracking, an enhanced set of contour-based areas like forest paths, coastlines, and power lines can be detected if the appropriate modifications and training data are available. In that direction, a higher resolution DVS and a neuromorphic chip with bigger capabilities could be deployed. Moreover, using more sophisticated SNNs ([44], [45], [46]) could provide even better perception results in more challenging environments, especially dynamic ones such as coastlines. Finally, utilizing more advanced event-based control strategies like the Event-Triggered Model Predictive Control (MPC) could further improve performance while keeping the power consumption sufficiently low.

Appendix A

FPGA Design

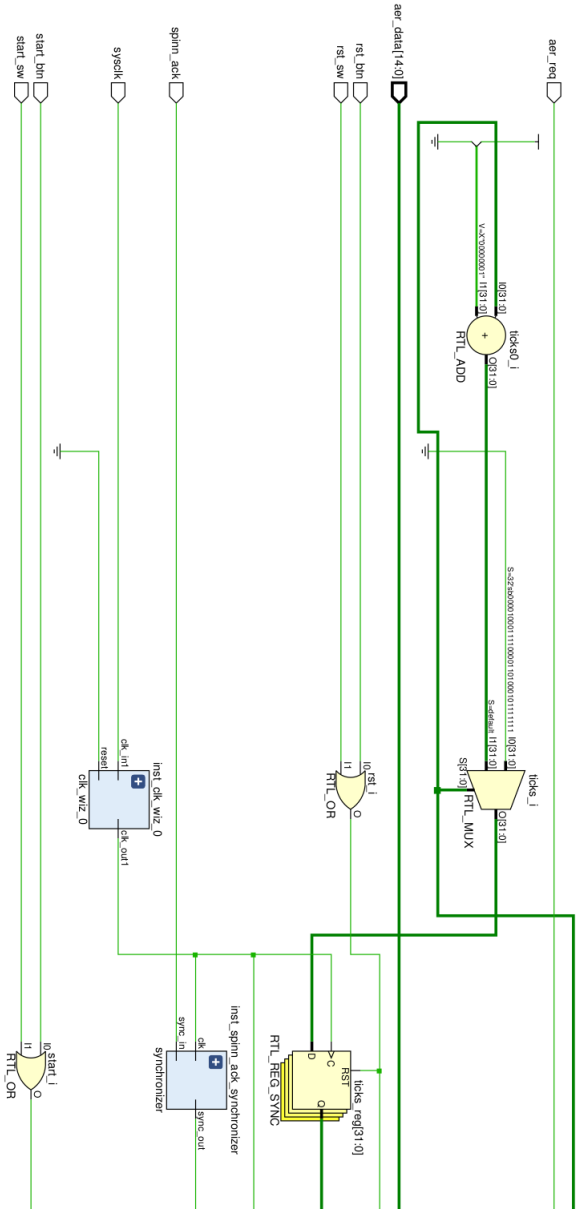


Figure A.1: RTL design of the DVS-SpiNN-3 Interface **(1/2)**.

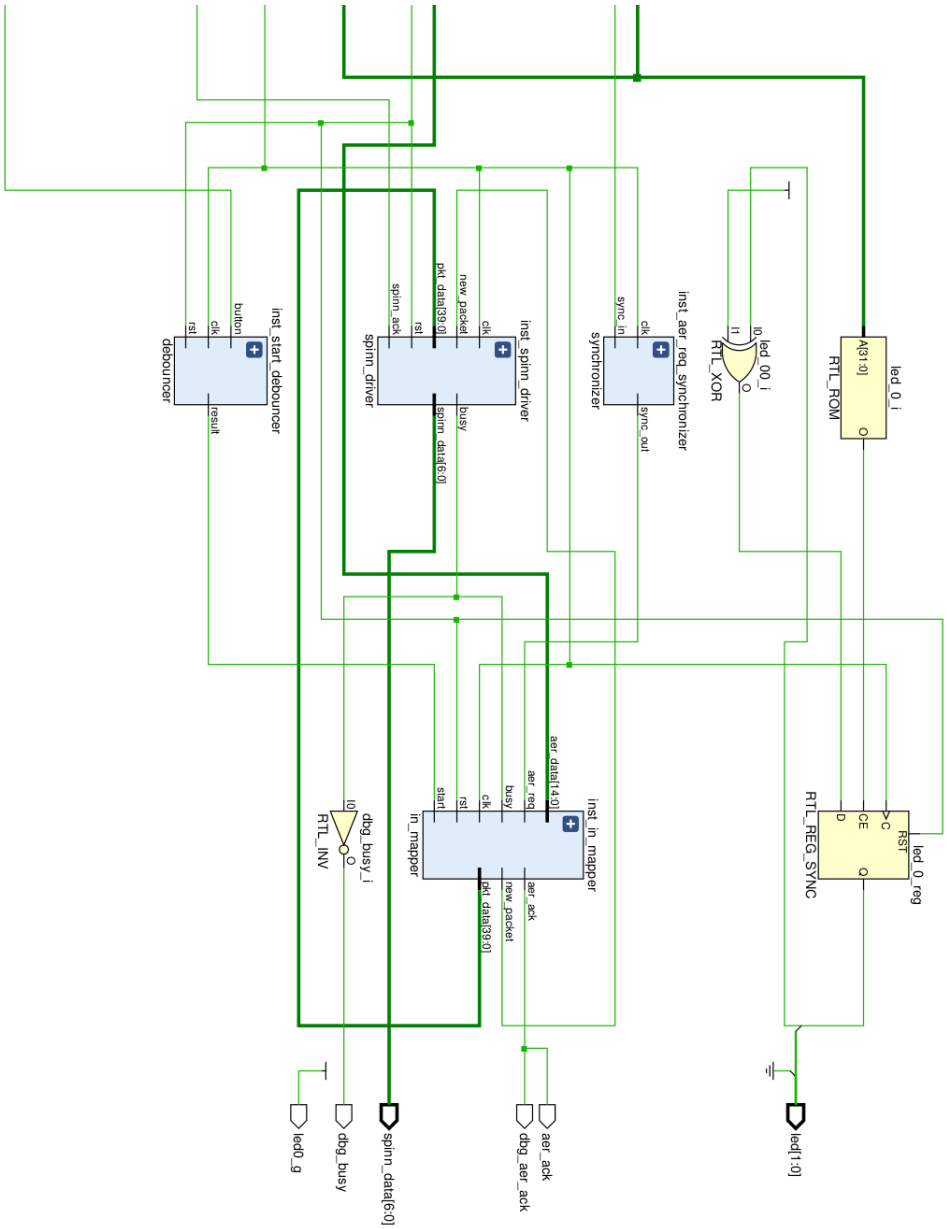


Figure A.2: RTL design of the DVS-SpiNN-3 Interface **(2/2)**.

Bibliography

[1] Guillermo Gallego et al. "Event-based vision: A survey". In: *IEEE transactions on pattern analysis and machine intelligence* 44.1 (2020), pp. 154–180.

[2] Paul A Merolla et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197 (2014), pp. 668–673.

[3] Ben Varkey Benjamin et al. "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716.

[4] Muhammad Mukaram Khan et al. "SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor". In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. Ieee. 2008, pp. 2849–2856.

[5] Steve Furber et al. "Overview of the SpiNNaker System Architecture". In: *Computers, IEEE Transactions on* 62 (Dec. 2013), pp. 2454–2467.

[6] Steve Furber et al. "The SpiNNaker project". In: *Proceedings of the IEEE* 102 (May 2014), pp. 652–665.

[7] Mike Davies et al. "Loihi: A neuromorphic manycore processor with on-chip learning". In: *Ieee Micro* 38.1 (2018), pp. 82–99.

[8] Ruoxiang Li et al. "Fa-harris: A fast and asynchronous corner detector for event cameras". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 6223–6229.

[9] Ignacio Alzugaray and Margarita Chli. "Asynchronous corner detection and tracking for event cameras in real time". In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3177–3184.

[10] A Gómez Eguíluz et al. "Asynchronous event-based line tracking for time-to-contact maneuvers in uas". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 5978–5985.

[11] Juan Pablo Rodríguez-Gómez et al. "Auto-tuned event-based perception scheme for intrusion monitoring with uas". In: *IEEE Access* 9 (2021), pp. 44840–44854.

[12] Elias Mueggler, Basil Huber, and Davide Scaramuzza. "Event-based, 6-DOF pose tracking for high-speed maneuvers". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 2761–2768.

[13] Yi Zhou, Guillermo Gallego, and Shaojie Shen. "Event-based stereo visual odometry". In: *IEEE Transactions on Robotics* 37.5 (2021), pp. 1433–1450.

[14] Sihao Sun et al. "Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 580–587.

[15] Anton Mitrokhin et al. "Event-based moving object detection and tracking". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–9.

[16] Bas J Pijnacker Hordijk, Kirk YW Scheper, and Guido CHE De Croon. "Vertical landing for micro air vehicles using event-based optical flow". In: *Journal of Field Robotics* 35.1 (2018), pp. 69–90.

[17] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. "Dynamic obstacle avoidance for quadrotors with event cameras". In: *Science Robotics* 5.40 (2020), eaaz9712.

[18] A Gómez Eguíluz et al. "Why fly blind? Event-based visual guidance for ornithopter robot flight". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 1958–1965.

[19] Ran Cheng, Khalid B Mirza, and Konstantin Nikolic. "Neuromorphic robotic platform with visual input, processor and actuator, based on spiking neural networks". In: *Applied System Innovation* 3.2 (2020), p. 28.

[20] Julien Dupeyroux et al. "Neuromorphic control for optic-flow-based landing of MAVs using the Loihi processor". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 96–102.

[21] Antonio Vitale et al. "Event-driven vision and control for UAVs on a neuromorphic chip". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 103–109.

[22] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[23] Christoph Posch et al. "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output". In: *Proceedings of the IEEE* 102 (Oct. 2014), pp. 1470–1484.

[24] K.A. Boahen. "Point-to-point connectivity between neuromorphic chips using address events". In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 47.5 (2000), pp. 416–434.

[25] P. Lichtsteiner and T. Delbruck. "A 64x64 aer logarithmic temporal derivative silicon retina". In: *Research in Microelectronics and Electronics*. 2005, pp. 202–205.

[26] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. "A 128× 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor". In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576.

[27] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. "A 128×128 120dB 30mW asynchronous vision sensor that responds to relative intensity change". In: vol. 43. Mar. 2006, pp. 2060–2069.

[28] Maria Teresa Serrano Gotarredona, Juan Antonio Lenero Bardallo, and Bernabe Linares Barranco. "A bioinspired 128x128 pixel dynamic-vision-sensor". In: *26th Conference on Design of Circuits and Integrated Systems (DCIS 2011)(2011), p 1-6*. 2011.

[29] Lei Deng et al. "Rethinking the performance comparison between SNNS and ANNS". In: *Neural Networks* 121 (2020), pp. 294–307. ISSN: 0893-6080.

[30] Timothée Masquelier, Rudy Guyonneau, and Simon J. Thorpe. "Leaky Integrate-and-Fire (LIF) neuron." In: (Feb. 2013).

[31] *Software for SpiNNaker*. URL: https://spinnakermanchester.github.io/.

[32] Steve Temple. *AppNote1 - SpiNN-3 Development Board*. 2011. URL: https://spinnakermanchester.github.io/docs/spinn-app-1.pdf.

[33] P. Hafliger. *CAVIAR - Hardware Interface Standards, Version 2.0, Deliverable D_WP7.1b*. 2003. URL: http://www2.imse-cnm.csic.es/caviar/download/ConsortiumStandards.pdf.

[34] Steve Temple. *AppNote7 - SpiNNaker Links*. 2012. URL: https://spinnakermanchester.github.io/docs/spinn-app-7.pdf.

[35] Luis A. Plana. *AppNote8 - Interfacing AER devices to SpiNNaker using an FPGA*. 2017. URL: https://spinnakermanchester.github.io/docs/spinn-app-8.pdf.

[36] Digilent. *CMOD A7-35T*. URL: https://digilent.com/reference/programmable-logic/cmod-a7/start.

[37] Sajjad Seifozzakerini et al. "Event-Based Hough Transform in a Spiking Neural Network for Multiple Line Detection and Tracking Using a Dynamic Vision Sensor." In: *BMVC*. Vol. 94. York, UK. 2016, pp. 1–12.

[38] Sajjad Seifozzakerini et al. "Hough transform implementation for event-based systems: Concepts and challenges". In: *Frontiers in computational neuroscience* 12 (2018), p. 103.

[39] François Chaumette and Seth Hutchinson. "Visual servo control. I. Basic approaches". In: *Robotics & Automation Magazine, IEEE* 13 (Jan. 2007), pp. 82–90.

[40] François Chaumette and Seth Hutchinson. "Visual servo control. II. Advanced approaches [Tutorial]". In: *Robotics & Automation Magazine, IEEE* 14 (Apr. 2007), pp. 109–118.

[41] OpenCV. *Camera Calibration and 3D Reconstruction*. URL: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html.

[42] Sotirios N. Aspragkathos, George C. Karras, and Kostas J. Kyriakopoulos. "A Hybrid Model and Data-Driven Vision-Based Framework for the Detection, Tracking and Surveillance of Dynamic Coastlines Using a Multirotor UAV". In: *Drones* 6.6 (2022).

[43] MAVLink Micro Air Vehicle Protocol. *MAVLink to ROS gateway with proxy for Ground Control Station*. 2013. URL: https://github.com/mavlink/mavros.

[44] Xavier Lagorce et al. "HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.7 (2017), pp. 1346–1359.

[45] Antoine Grimaldi et al. "A robust event-driven approach to always-on object recognition". In: (Jan. 2022).

[46] Kinjal Patel et al. "A Spiking Neural Network for Image Segmentation". In: (2021).