Neuro-evolution learned Neuromorphic Control for a Vision-based 3D Landing

E. Lodder

TU Delft

Neuro-evolution learned Neuromorphic Control for a Vision-based 3D Landing

Thesis report

by

E. Lodder

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on July 7, 2023 at 09:30

Thesis committee:

Chair:	Dr. G.C.H.E. de Croon
Supervisors:	Dr. G.C.H.E. de Croon
	S. Stroobants
Examiner:	Dr. C. de Wagter
External examiner:	Dr. C.J.M. Verhoeven
Place:	Faculty of Aerospace Engineering, Delft
Student number:	4300548

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

Faculty of Aerospace Engineering · Delft University of Technology

Contents

Nomenclature

List of Abbreviations

AELIF	Adaptive Exponential LIF
AER	Address-Event Representation
AI	Artificial Intelligence
ANN	Artificial Neural Network
ATIS	Asynchronous Time Based Image Sensor
CH	Centrifugal Horizontal
CMA-ES	Covariance Matrix Adaptation ES
CNN	Convolutional Neural Network
CPPN	Compositional Pattern Producing Networks
DAVIS	Dynamic and Active Pixel Vision Sensor
dCH	dorsal CH
DE	Differential Evolution
DLN	Deep Learning Networks
DNN	Deep Neural Networks
DVS	Dynamic Vision Sensor
EA	Evolutionary Algorithms
EB	event-based
eDVS	embedded DVS
EINDI	Extended INDI
EMD	Elementary Movement Detectors
ES	Evolutionary Systems
ESN	Echo State Network
FAST	Features from Accelerated Segment Test
FoE	Focus of Expansion
GPS	Global Positioning System
HS	Horizontal System
HSE	HS Equatorial
HSN	HS North
IF	Integrate-and-Fire
IMU	Inertial Measurement Unit
INDI	Incremental NDI
INS	Inertial Navigation Sensors
LIF	Leaky Integrate-and-Fire
LPTC	Lobula Plate Tangential Cells
LQE	Linear Quadratic Estimator
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
LSM	Liquid State Machine
LTD	long term depression
LTP	long term potation
MAV	Micro Air Vehicle
MDGL	Multidigraph Learning Rule
ML	Machine Learning
MLP	Multilayer Perceptron
MOEA	Multi Objective Evolutionary Algorithms
MOEA/D	Multi-Objective Evolutionary Algorithm based on decomposition
MPC	Model Predictive Control
NDI	Non-linear Dynamic Inversion
NE	Neuromorphic Engineering
NES	Natural Evolution Strategies
NL-MDGL	Non-Local MDGL
NMPC	Non-Linear MPC
NN	Neural Network
NSGA	Non-dominated Sorting Genetic Algorithm

NSGA-II Non-dominated Sorting Genetic Algorithm II

ODE Ordinary Differential Equations

OF Optical Flow

PID Proportional Integral Derivative

RC Reservoir Computing

RL Reinforcement Learning

RNN Recurrent Neural Network

RTRL Real-TimeRecurrent Learning

SG Savitzky-Golay

SLAYER Spike Layer Error Reassignment in Time

SMC Sliding Mode Control

SNN Spiking Neural Networks

SPEA Strength Pareto Evolutionary Algorithm

SRM Spike Response Model

STDP spike timing dependent plasticity

TRTRL Truncated Real-Time Recurrent Learning

TTC time-to-contact

UAV Unmanned Aerial Vehicle

vCH ventral CH

VS Vertical System

List of Figures

List of Tables

1

Introduction

Micro Air Vehicles (MAV) are small and light air vehicle systems that, due to their scale, are beneficial for search, exploration and observation applications. For these vehicles to be self-reliant and independent of external navigational aid, all computations regarding navigation, trajectory planning and control require on board calculations and environmental awareness. Sequentially, on board systems are constrained by scale and power consumption. Making sensors and processors smaller and more powerful will allow MAVs to decrease further in size or adopt more sophisticated perception and computation possibilities. Reducing the scale of MAVs is beneficial for such missions since size is less of a constraint for narrow or indoor environments and build/operating costs in general are lower for smaller vehicles.

Insects and birds are commonly regarded as the biological counterpart of MAV's. Flying insects essentially establish a benchmark considering their size, and navigational and control capacities based on visual input. Precise object avoidance and agile manoeuvring are features that are realized, though, with low computational power. The level of intelligence (perception and algorithmic complexity) of MAV's is dictated by sensors and on board computing power. The processing units on board most MAV's possess a conventional Von Neumann architecture [1] that must grow in size to increase computational capabilities and processing power. A size and power usage increment is undesirable for MAV's that owe many beneficial properties to their small size.

The previous is not only regarded as a consideration for robotic applications, but generally speaking, might be troublesome as the efficiency of current hardware can not be increased [2]. As future (robotic) applications will demand more computing power, the only resolution will be increasing the number of processing cores and thus inevitably, size. Another downside of conventional chips are the limited possibilities for parallelism while operating, which renders to continuously executing and therefore power hungry cores. These cores are continuously engaged, even when less computational capabilities are required. Different alternatives opposed to conventional computing methods for robotic applications are searched for. A crucial source of inspiration are the nervous systems and brains of animals and humans. Neuromorphic engineering resembles a realistic neuro-biological approach that mimics the computational principles observed in animal and human nervous systems. By structuring the electronic circuits as our nervous system, biological cognitive abilities can be approached. A key factor that can be attributed to these improvements is the asynchronous processing capabilities of such systems with electronic analog circuits. Such devices are affiliated with neuromorphic hardware, allowing neuromorphic computing. Recent developments indicate an increasing interest in neuromorphic computing because of the decreased, in several magnitudes, computational power and increased arithmetic capabilities since neuromorphic hardware allow cognitive alike parallelism. A key difference between the on-chip logic and memory storage functions, currently present on most chips and neuromorphic hardware, is the representation of information. Like nervous systems, neuromorphic hardware allows binary, sparse and asynchronous input which is hard to interpret for traditional arithmetic on-chip computations. Another key difference lies in the execution, since neuromorphic chips are driven by event-based processing. The latter means that neurons will only operate and require power when the neuron has been engaged. Next to event-based processing, neuromorphic components can operate in parallel since the neuronal efficacies are independent of each other. Additionally, as only simple non-information conveying impulses are transmitted, processing speeds are faster. Since spikes over electric analog circuits are means of communication, almost no time delays exist while processing.

Not only has event-based processing been an inspiration for computation, but for sensing as well. Sensors that mimic the dynamic processing of motion of the visual field have been developed and are described as a Dynamic Vision Sensor (DVS) [3]. By switching from frame-based image data to (almost) continuous vision data with a spatial and temporal component, redundant scene information is avoided. As the DVS only produces an output when a (small) brightness changes observed at pixel level, vision information is delivered efficiently, which makes such neuromorphic sensors suitable for robotics. A downside of a DVS is the lack of static scene information, such as absolute brightness, making it more difficult to interpret for current machine vision algorithms.

With an increase in efforts aspiring neuromorphic engineering, the algorithms most fit to function on neuromorphic hardware are increasingly being applied to robotics to improve the practicality and applicability. Also known as the third generation AI [4], Spiking Neural Networks (SNN) possess the same traits and dynamics as explained before, namely working in asynchronous and sparse conditions. The spikes generated in a SNN have a spatial and temporal component, and the neurons communicate with each other through synapses over which impulses are being sent. A key aspect of a SNN model lies in the dynamics of a neuron, which can differ, but generally speaking, represent a differential equation. The latter encompasses the importance of the timing of spikes, justifying the increased complexity SNN's can process and yield for any type of application. Evidently, the previous properties given of neuronal dynamics are expressed in a general sense, but more on the exact modelling of dynamics, networks and cognitive abilities can be found in [5].

A key aspect of implementing neuromorphic control is the learning algorithm. The learning algorithm is responsible for tuning the decision parameters, such as synapse weights and decay values. The learning algorithm should ensure the most optimal output by means of the past and current input. Although learning SNN's for neuromorphic control is gaining traction for robotics, a clear leading technique does not exist. Many control applications apply a significant simplification, which decreases the general usability of the learning scheme used. For example, simulated environments would often be used, ignoring the non-linearities present in the real environment. Other real world control applications of SNN involve transforming conventional control loop mechanisms, such as a Proportional Integral Derivative (PID) controller, into a framework fit for a SNN. This would translate to using specific encoding and decoding scheme's, where the neuron properties in a SNN would be altered to perform basic arithmetic operations [6]. Although using a SNN in closed loop continuous control is achieved, the true non-linear performances empowered by the neuronal dynamics are not reached to the fullest.

2

Research Question

From the introduction, we have learned about the advantageous neuromorphic properties and why MAV's might benefit from them. The aim is to demonstrate that an end-to-end event based processing system can be designed for carrying out a complex flight control task on MAVs. The motivation is to show that the controller can capture non-linear and real world environment dynamics, showcasing the advantageous spatio-temporal processing capabilities of neuromorphic hardware. This allows us to deduce the main research question:

Can a Spiking Neural Network be evolved, using Evolutionary Strategies, to map Event-Based Optical Flow observables to motor commands for a Micro Air Vehicle, proving biologically plausible visuomotor characteristics of insects, to perform a 3D controlled real-world landing using Neuromorphic Hardware ?

The research objective is therefore twofold:

1. How to design a neuromorphic controller ensuring appropriate visuomotor coordination for MAV motor control
2. Implement a full event-based end-to-end processing system

Three main themes can be identified according to the research question and objectives. These roughly cover the perception, processing and robotic application part of the experiment. The contribution will consist of the following three aspirations: Determine suitable spike-based visual observables for flight control, evolve a SNN for generating optimal motor commands and finally, implement the system on neuromorphic hardware for a real-world 3D controlled landing. According to these contributions, the following research sub-questions are formulated:

1. *How can we acquire visual features from event-based sensors for vision based navigation that are suitable for identifying ego-motion and navigation purposes ?*
2. *How can we evolve a SNN-based motor controller using visionary inputs ?*
3. *How to implement an end-to-end event-based system on a MAV for a real-world 3D controlled landing ?*

2.1. Structure of work

This thesis report is structured as follows: The first part will include the scientific paper as well as the appendices belonging to the paper. Chapter 4 displays the current landscape of landing controllers for MAV's. By understanding current methods and their properties, we will be able to comprehend important features of control systems and compare performance. The next chapter analyses the fundamentals of vision based flight. Here, different optical flow estimation techniques are examined, as well as the corresponding biologically inspired flight control strategies. Event-based vision will be discussed in the second part of this chapter. Following is chapter 6, which discuss how insect interpret and utilize vision for different manoeuvres and navigation purposes. The organization of the receptive field shows the directional sensitivity of perception. Chapter 7 will explain different evolutionary algorithms and their properties. Next to the fundamentals of evolutionary frameworks, more recent developments, that take into account current reward based learning schemes, will be discussed. Chapter 7 studies the neuromorphic engineering

domain for robotic applications. By having a look at modelling, hardware and applications, an understanding is formed of the learning and implementation challenges concerning neuromorphic computing. The last chapter discusses the findings from the literature study. Finally, this report is concluded with a conclusion and recommendations with possible guidelines for future continuation.

Part I

Scientific Article

Neuro-evolution learned Neuromorphic Control for a Vision-based 3D Landing

Erwin Lodder[1], Stein Stroobants[2], Guido C.H.E. de Croon[2]

Neuromorphic control is a biologically inspired processing method that uses Spiking Neural Networks (SNN) to simulate the computational principles observed in mammal nervous systems. Micro Air Vehicles (MAV) are small and light air vehicle systems of which the level of intelligence is dictated by sensors, algorithmic complexity and onboard computing power. Implementing flight control systems with SNNs would allow us to pursue an end-to-end event-based processing sytem that decreases scale and power usage, which can substantially increase the artificial intelligence of MAVs. With the rise of available neuromorphic sensors and processors, one would expect to find many robotic applications. However, learning and the reality gap impede robotic applications. This work shows that a single SNN controller can be evolved for 3D flight control (thrust, roll and yaw). By learning an Optical Flow (OF) based flight control strategy, it is proven that a neuromorphically controlled MAV can take-off, cruise and land on a target with great precision ($x_\epsilon, y_\epsilon < 0.02m$ & $V_z < 0.05ms^{-1}$). The extracted ventral flow in the longitudinal axis controls motion in the for x and z in the xz plane. By means of evolution, not only are the SNN network parameters evolved, the topology is modified during training as well. Instead of using a pre-defined fully connected Multi Layer Perceptron (MLP) like architecture, a randomly connected and sparse network is custom built and learned.

I. Introduction

Micro Air Vehicles (MAVs) represent a robotic embodiment of insects that pursue the same autonomous manoeuvrability and navigational capabilities that flying insects have. For MAVs to be self-reliant and independent of external navigational aid, all computations regarding navigation, trajectory planning and control require onboard calculations. Not only can flying insects perform high speed visuomotor computations, they do so with highly limited computational resources. While most autonomous MAV applications require external navigational aid (GPS) or multiple sensors complementing the onboard camera [1], for flying insects however, determination of ego motion heavily depends on visual information. In order to keep the payload as small as possible, it is desirable to use low weight and low power consuming sensory equipment and processors to fit with MAVs' constraints.Neuromorphic engineering more closely approximates the computational principles observed in animal and human nervous systems. By structuring the electronic circuits as our nervous system, biological cognitive abilities can be approached. Also known as the third generation networks [2], Spiking Neural Networks (SNN) possess the same traits and dynamics as explained before, namely, working in asynchronous and sparse conditions. The spikes propagated through an SNN have a spatial and temporal component, and the neurons communicate with each other through synapses. Neuromorphic control refers to artificially constructed networks that transfer these event-based processing characteristics to control systems. The energy efficient and high speed processing characteristics will allow insect-scale MAV applications.

While SNNs possess beneficial characteristics for robotic applications, the number of real-world experiments is relatively low. This is due to the non-linear learning traits and gap between simulation frameworks and real world experiments. Sparse and asynchronous features do not allow well-known gradient learning to work as efficiently as for traditional Artificial Neural Networks (ANN). In order to deal with the binary activation functions of SNN's, surrogate gradients [3] are typically used to enable learning with backpropagation. Additionally, most deployed SNN architectures are similar to fully connected MLPs, since Convolutional Neural Networks (CNN) [4] and MLPs are the most commonly used architectures for ANNs. However, bio-inspired SNNs are sparse and randomly connected. The dynamic and recurrent functioning of SNNs are due to recurrent connections, the membrane potential of each neuron and spikes that are carried across multiple network layers. Unfortunately, the fixed architecture of dense feed forward ANNs, as simulated in frameworks like PyTorch [5], do not allow sparse and layer independent connections to exist while these architectures are common in bio inspired SNNs.

The current state-of-the-art in neuromorphic control for MAV platforms contains different kind of SNN applications. The authors in [6] present a SNN based PID controller for a one DOF control task of an UAV. The architecture of a neuromorphic chip is utilized to perform arithmetic calculations, though no learning is involved. The authors in [7] have evolved SNNs for controlling a pure vertical landing of a MAV. The closed loop SNN controller uses the extracted divergence to control motor thrust for a landing. The attitude estimation network learned in [8] is able to determine flight control angles while only receiving raw IMU data. These previous applications have shown the potential that fully embedded neuromorphic control systems have on MAVs.

The compound eye of insects possesses event-based vision of which the output, in contrast to traditional cameras, is dependent on motion in the scene viewed. The output of event-based cameras, such as a Dynamic Vision Sensor (DVS), has a temporal and spatial component, whereas the temporal property for frame-based cameras is fixed. Spikes are encoded from visual stimuli by light-sensitive cells that react to viewed brightness shifts in the environment, with a latency close to microseconds. Asynchronous log intensity changes are recorded at pixel level and therefore possess a spatial component. Neuromorphic sensors have shown that this event-based processing is fast and can encode visual information at a low granular level [9]. The processing of these spikes is characterized by sparse and asynchronous properties, as the activations of single pixels across the sensor are sent on independently.

[0]1 Student 2 Supervisor

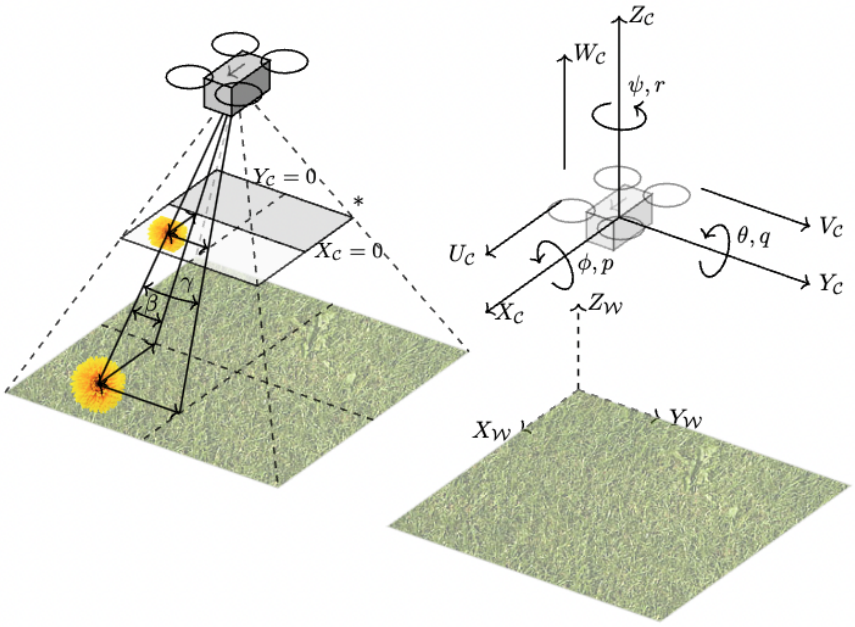


Figure 1. Both figures display the same base of a ground surface and MAV. The left image shows the according geometries as viewed underneath the camera, while the right image shows the coordinate frames. β determines the pitch setting of the camera and thus the MAV. This pitch setting is not influenced by the SNN controller and is set outside the control loop. γ encodes the heading error. The surface between the ground surface and MAV represents the camera sensor. The grey area of this surface (with asterisk) is not used for flow u determination. In the right image, two reference frames are present, namely of the World frame $\mathcal{W}$ and Camera frame C. The motion of the camera is influenced by translational velocities (U_C, V_C, W_C) and rotational rates (p, q, r). The surface nearest to the asterisk represents the camera sensor.

Flying insects solve odometry, obstacle avoidance and navigational problems by reacting to relative movements observed in the field of view caused by self-motion. These patterns in visual motion are referred to as Optical Flow (OF). OF is defined as the apparent motion between the observer and scene and depends on the distance between them. OF does not allow absolute magnitudes to be determined and can not disentangle velocity and distance [10]. Insects use OF for different flight strategies. For example, during cruise and landing, it is shown that insects maintain a constant OF.

Robotic applications with OF based control either are limited by processing power needed for frame-based OF extraction, or low number of pixels on an Elementary Motion Detection (EMD) constraining more complex applications. Both limitations only permit one or two-dimensional control tasks to be executed. While the level of visual processing efficiency, in terms of speed and power, of insects will for now be unreachable, MAVs would benefit of a DVS and neuromorphic processors since the sensory equipment fits the control systems functioning which can be deployed at a small scale while consuming low power. This work will show that OF extracted from only visual information, can be used by a single SNN controller for 3D flight control. By using the same OF control strategy, take-off, cruise and landing can be performed.

The contribution of this work is two-fold: (1) Using only visual cues obtained from onboard cameras for attitude commands, a flight including take off, cruise and landing is performed. Since in nature, the longitudinal plane is regarded as the most important direction for covering distance, a heading-following control system is introduced, for which a single SNN controller is learned that controls thrust, roll and yaw. It is shown that using noise settings extracted from real world experiments, precise flight control is achieved.

(2) A flight controller is evolved using NEAT-SNN. By means of parameter and topology evolutions, an SNN is constructed. The previous allows the most common simulation frameworks to be used while building and learning more efficient sparse networks. Such method allows random sparse networks to be created that, in terms of topology, are closer to the networks found in the brain and nervous systems than MLPs. NEAT-SNN randomly places neurons and synapses while mutating the weights and parameters of existing synapses and neurons. An algorithm is developed that converts adjacency lists created by NEAT to PyTorch models without comprising the efficacy of spiking neurons.

II. Methodology

A. Determining ego-motion from optical flow

In order to operate an OF based flight control strategy, the OF parameters must be defined. The model used for interpreting optical flow is based on [11], which assumes that the retina or sensor can be viewed as a plane and the camera's aperture is viewed as a pinhole-point. The position of a point in both the World frame W and Camera frame C is defined by $(X_{\mathcal{W}}, Y_{\mathcal{W}}, Z_{\mathcal{W}})$ and, (X_C, Y_C, Z_C) respectively. (U_C, V_C, W_C) are the corresponding velocity components in m/s of a point in the C frame. The Euler angles (ϕ, θ, ψ) express the orientation of frame C with respect

to $\mathcal{W}$ and represent roll, pitch and yaw respectively. Additionally, (p, q, r) are the corresponding rotational rates. Furthermore, the camera pixel coordinates are (x, y) and their velocities, which represent OF components, denote (u, v) in rad/s. The exact labels and axis can be found in Figure 1.

The projection of an arbitrary point in the reference frame C onto the image plane is denoted by $(x, y) = (\frac{X_C}{Z_C}, \frac{Y_C}{Z_C})$. Due to the observer's motion, this point moves with velocity (u, v) across the image plane. Utilizing the time derivative of points on the image plane, the OF of a point can be expressed into translational and rotational components :

$$u = -\frac{U_C}{Z_C} - U_C\frac{W_C}{Z_C^2} = (-\frac{U_C}{Z_C} - q + ry) - x(\frac{W_C}{Z_C} - py + qx) \quad (1)$$

$$v = -\frac{V_C}{Z_C} - V_C\frac{W_C}{Z_C^2} = (-\frac{V_C}{Z_C} - rx + p) - y(\frac{W_C}{Z_C} + qx - py) \quad (2)$$

For general applications of OF, it is beneficial to separately measure the rotational rates (p, q, r). Including these rates in equations 1 and 2, will only let translational flows remain as the observed states in (u, v). This is called de-rotation as is common for OF-based control [12], where ego-rotational movements are isolated from determining (u, v) . For this work, flows caused by vertical speed, are not considered in ventral flow v. Furthermore, the observed flow u controls both z and x in the longitudinal (x, z) plane of the Camera frame C. Since the flow is measured symmetrically around the $Y_C = 0$ axis, all terms containing y in eq. 1 can be cancelled out. This also applies to v in eq. 2 though measured around the $X_C = 0$ axis. Additionally, pitch is not considered a controlled variable, though, pitch rate is still considered during transition phases from take-off to cruise, and cruise to landing. This leads to

$$u \approx (-\frac{U_c}{Z_c} - x\frac{W_c}{Z_c} - q - qx^2), \quad (3)$$

$$v \approx (-\frac{V_c}{Z_c} - p - py^2), \quad (4)$$

where $q = 0$ during cruise. The ventral flow u observed in the shaded area on the sensor in Figure 1 contains flows that, for vertical and forward speed increments, have an opposite sign. It is desirable that these coupled variables have opposite signs in u. The flows detected by pixels in the unshaded area of the sensor in 1, are not considered for OF computations.

B. 3D visual flight strategy

The evolved 3D flight controller is a high level controller that does not concern attitude control. Therefore, the controller outputs a control signal for thrust T, roll angle ϕ and yaw angle ψ. Thrust controls motion in the xz plane of the Camera frame C, and roll and yaw control the heading-following control of the MAV. The input signal consists of a ventral flows u and v and course error γ. It is assumed that the desired heading is known, which can either be calculated from a target (flower in Figure 1 or while following a ground path.

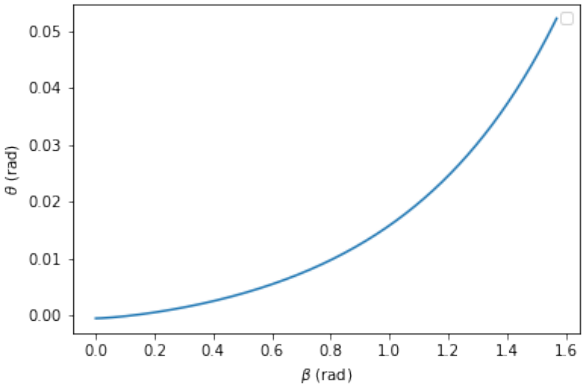


Figure 2. The pitch setting is dictated by β, which depends on the height and distance between the MAV and target (Figure 1).

Since it is hard to disentangle the cause of flow increments in u by thrust T or pitch θ adjustments, the pitch is fixed as in [13]. As explained in [10], OF can not disentangle distance and velocity, which makes it difficult to control T and from the same flow. Pitch is therefore determined according to the distance and height between the MAV and target, which translates to an angle β. The larger the angle between the centreline ($X_C = 0$) of the sensor and target as viewed on the sensor, the greater the pitch. The angle β is not encoded in the input layer, but adjusted for outside the control loop. The exact pitch setting according to angle β can be found in Figure 2. During take-off and cruise, it is assumed that a ground track is followed without an exact target, wherefore pitch can be set to an arbitrary setting. The pitch setting can either be increased over time during take-off or set at a pre-determined value. The heading error is determined by a separate target or line on the ground surface that the controller must follow. For take-off and cruise, a line is followed while for landing, a target on the ground surface has an established position. Here, the heading error γ is defined as the angle between the longitudinal centreline of the MAV and target.

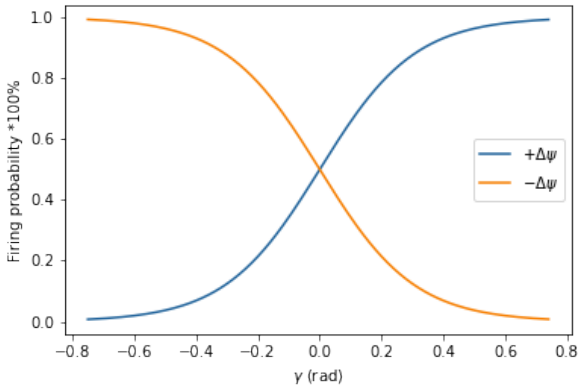


Figure 3. The graphs viewed represent a separate encoding neuron. Both neurons receive a similar spiking probability if the camera's centreline $X_C = 0$ has the same heading as the target (Figure 1).

In case of the line-following control scenario, the angle between the centreline of the sensor at ($Y_C = 0$) and the line viewed at the top of the sensor, depicts the heading error γ. This error signal is encoded by two neurons. Each encoding neuron represents a different error sign, whereas the spiking frequency represents the angle magnitude. Figure 3 shows the firing probability of both nodes according to the heading error magnitude. If the heading of the MAV is similar to the target's bearing, both encoding neurons fire at $P(\gamma = 0) = 0.5$.

C. SNN properties

The synapses in networks that connect neurons have a certain weight w_{ij}. The membrane potential $u_i(t)$ in each neuron is influenced proportionally by the weight from the synapse of which a spike is received. If no inputs are received, $u_i(t)$ will decay to a resting potential u_{rest}. Though, if $u_i(t)$ reaches a certain threshold θ_i, a spike s_i is emitted after which $u_i(t)$ will reset to u_{rest}. The synapses are static, and the biologically inspired neuron model used for this work is the leaky integrate-and-fire (LIF) model. After discretizing the differential equation with forward Euler, the membrane potential $u_i(t)$ is modelled as,

$$u_i(t) = u_i(t - \Delta t)\tau_{u_i} + i_i(t) \tag{5}$$

where it is assumed that $u_{rest} = 0$, τ_{u_i} is the membrane decay factor and i_i the incoming spike function of neuron i, which multiplies the synapse weights by spikes. $i_i(t) = \Sigma_j w_{ij} s_j(t)$.

Since SNNs only can interpret binary spikes as input, the continuous input signal must be converted from a real-valued signal to binary spikes (encoding). The same accounts for the output, as most systems require continuous real-valued signals (decoding). For the encoding, a pair of spiking neurons is used to address both a positive and negative value of an input signal a_i. One neuron of each pair of encoding neurons spikes at each time step. The magnitude of the signal is multiplied by the weights of synapses in the first layer before reaching the first layer of hidden neurons. This means all values are positive floating point values. The sign of the encoding signal determines which neuron of each encoding pair is fired 7. All values that are fed into the network are positive, whereas the signal sign translates to a spatial component. To avoid spikes being fired that have a larger magnitude than $|a_i| > 1$, another neuron is added that spikes the remainder of $|a_i| - 1$ when the input signal exceeds magnitude 1.

$$s_j^+ = |a_i| \quad \text{if } a_i > 0 \quad \text{else} \quad 0, \tag{6}$$

$$s_j^- = |a_i| \quad \text{if } a_i < 0 \quad \text{else} \quad 0 \tag{7}$$

For decoding the binary spikes, a non-spiking LIF is adapted, which essentially acts as a low pass filter. Every real valued output signal $X_i(t)$ receives a pair of output neurons. Again, the pair of neurons address positive and negative values. Both of the decoding neuron's membrane potential only can attain a positive value. However, to ensure that the system can output a broad range of positive and negative control signals, the membrane potential value of the negative neuron is subtracted from the positive neuron.

$$X_i(t) = X_{i_+}(t - \Delta t)\tau_{x_{i_+}} + i_i(t) - (X_{i\_}(t - \Delta t)\tau_{x_{i\_}} + i_i(t)$$

D. NEAT-SNN

NeuroEvolution of Augmenting Topologies (NEAT) is an Evolutionary Algorithm (EA) that evolves network topologies along synapse weights [14]. This framework allows sufficient exploitation and exploration of networks by a topology cross-over mechanism. The networks are categorized according to architecture properties into so-called species. By separating the evolutionary process of each species, structural innovations are protected. SNNs would benefit from such optimisation schemes, since sparse networks with random connections that are closer to bio inspired networks can be built. Since a non-continuous activation function is present SNN's neurons, a higher probability for SNNs in comparison to ANNs exist, that cross-overs produce over exciting or inactive neurons when two separately evolved SNNs are combined. Due to these possible signal propagation discontinuities present in SNNs when creating cross-overs, cross-overs are unfavourable. The maintenance mechanism of species is discarded, since the segmentation measure introduced by NEAT, that measures compatibility, becomes less applicable to SNNs. This means that individuals are only evaluated and compared to other individuals in a single species if new neurons or synapses improve performance.

In order to use NEAT for SNNs, modifications are required. The library NEAT-SNN we created evolves SNNs and can simulate SNNs in an MLP fashion. Where NEAT would formulate graph networks as separate genes, PyTorch and other common frameworks use fixed architectures. As most real-world applications require fixed architecture network formulations, it is key to be able to simulate NEAT networks in these fixed architecture frameworks. The reason for not using NEAT's own network simulation method is that most networks that are implemented on neuromorphic hardware, benefit from hierarchical top down networks with such fixed architecture [15] [16]. Furthermore, common frameworks like Pytorch have a large library's in where custom learning algorithms can be built and multiple custom neuron activation functions can exist. Besides the hidden states in neurons, further external memory is not necessary, as the synapse and neuron are updated online. This makes implementation on neuromorphic hardware easier. True recurrence can not be achieved, since these fixed architectures do not allow sequentially independent connections to exist. Therefore, the only recurrent property the evolved SNNs will possess is the membrane potential u_i, that is carried across multiple layers. Though, as will be shown later, neurons that are located in the same sequential layer can be connected. In such fashion, with skip connections, a neuron activation can be carried across multiple layers while other network activity is not disturbed. This simulates sparse connections that can span multiple layers. It is important to note that not every synapse can be created for the network to remain a feed forward network. NetworkX [17] is used to assess whether a synapse creation violates this feed forward property. Another important feature of such networks is that the architecture does not possess any symmetrical properties.

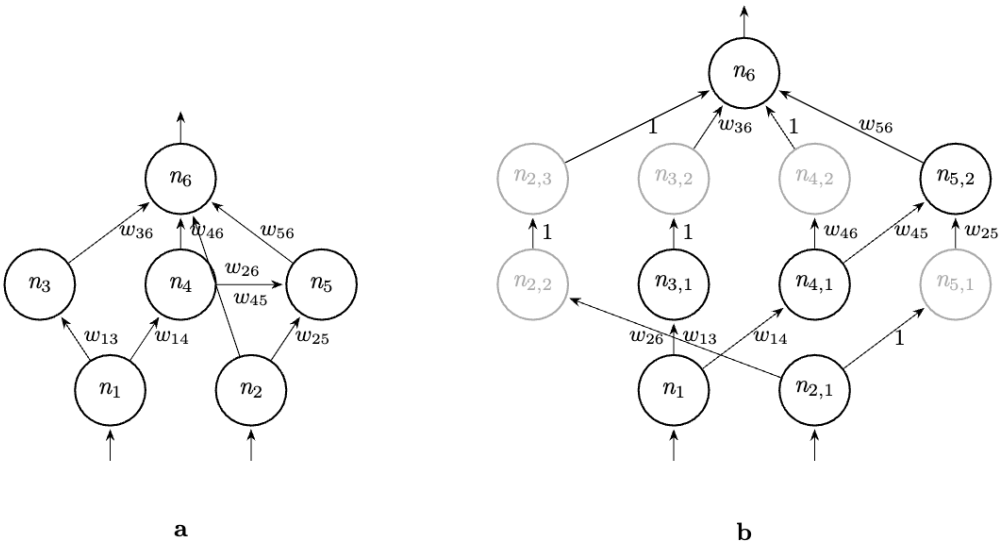


Figure 4. a) A randomly generated sparse network is shown. Neurons n_i are connected by synapses with weights w_{ij}. It is seen that synapse with weight w_{45} links two neurons in the same layer while w_{26} skips a layer. b The same network, though in an MLP format. Both networks share the same output behaviour, but simulate a SNN differently. The fixed architecture simulation frameworks require the weights between two sequential layers to be given. Here, other than for the weight specified, all weights are 0. A weight of 1 is given to neurons that propagate a spike across sequential layers. As for synapse weights, decay α_i and threshold θ_i can theoretically be chosen at random, as long as the neuron emits a spike when a spike is received. The neuron's parameters are chosen as such, that the neuron essentially operates as a synapse. The non-transparent nodes indicate the original neurons with the same neuron parameters as in network a.

Figure 4 shows the required transformation from a randomly generated sparse network to a fixed architecture MLP while maintaining the same efficacy. The transparent nodes indicate the additional required neurons for the MLP format. The neurons that bridge different layers, by means of a skip connection, carry an arbitrary weight, though the neuron must spike to propagate a signal. An algorithm has been created that places the according neurons and weights in the fixed architecture referred to earlier. The previous means that neurons must be placed in the correct layer, and the synaptic weights must be filled in an array that suffices as a placeholder of weights located between two consecutive layers. Transforming a sparse network to an MLP requires custom skip connections, as some connections span multiple sequential layers. A sparse network is converted to a fully connected MLP by solving the neuron placement problem. The algorithm first finds the longest 'route' (chain of neurons) between the input layer and output layer. As the longitudinal positions of these neurons are certain, neurons that are not in the longest route but are connected to these neurons, can be placed accordingly. The longest chain of neurons determines the number of layers in a network. Multiple solutions of neuron placement configurations exist, though the configuration with the least amount of skip connections, keeping the network as small as possible, is preferred. Since in an MLP the neurons of layer l are connected to every neuron in layer $l-1$, some connections receive weight 0. Iteratively, all neurons can be placed. The order of neurons in a layer does not matter, since there exists a placeholder connection between every neuron in 2 sequential layers. The scale or network size does not affect the workings, however more iterations are required for 'placing' neurons.

Table 1. NEAT parameters

Parameter	Value
Population	50
WeightMutationRate	0.3
ResetWeightRate	0.075
AddGeneMutationRate	0.43
AddNodeMutationRate	0.25
DecayMutationRate	0.2
ThresholdMutationRate	0.2
ResetDecayRate	0.05
ResetThresholdRate	0.075

E. Learning SNNs

Each evolution begins with a randomly initialized sparse network. At the start, without any hidden layers, the network is expanded up to a certain number of hidden neurons and synapses. From there, the evolution starts and network parameters (weights and neuron parameters) along topology are mutated every generation and optimized according to the control task. As explained in the previous section, cross-overs are not favourable due to possible discontinuities present in SNN's neurons. Although NEAT maintains fundamentally different networks in species, here, the networks only differ a couple of mutations from each other. This means that the topology of individuals in the same generation do not differ significantly. This also indicates that any topology can be learned as appropriate/required synapses will be added according to the control task.

A stochastic simulation environment, which will be elabo-

rated further upon in the next section, is used for the objective evaluation. In the environment, a pre-determined ground path is created that the controller should follow. Instead of a straight line, a more challenging sine wave $Y_{\mathcal{W}} = 5 \cdot \sin(0.5 X_{\mathcal{W}})$ marks the ground path. In order to challenge the controller in maintaining a constant u, Perlin noise [18] is added to the ground surface. Perlin noise is regarded as a realistic terrain generator and challenges the evolved controller even further, making it less prone to failure on flat surfaces. An example of a terrain profile can be found in the Appendix 15.

For training purposes, a function calculates the appropriate heading for the MAV to follow. This heading depends on the location of the MAV relative to the ground track. Would the MAV deviate from the path, will heading signal γ compensate for the deviation in the following heading commands to prevent the MAV drifting. The flight path is heavily constrained by speed and position requirements. In order to reach the end of the path, the MAV must fly through a maze with certain attitudes. The nearer the MAV is to the end of the path, the higher the reward. Thus, we are sure that the controller will only have reached the end of the environment according to imposed flight characteristics.

As viewed in equation 8, this aforementioned term dictates the reward signal, since the most reward is gained from successfully completing the objective of reaching the end of the environment. The intended trajectory has different constraints that cap or get rid of undesirable behaviour from the controller. Once the MAV has reached the end of the environment, the heading and OF control will be improved, though, all imposed constraints on the flight performance remain. Since the environment is highly stochastic, 50% of a generation's individuals, are considered for the next generation. To make sure learning does not converge too much in the beginning, these individuals are evaluated again in the next generation.

$$r_{penalty} = \text{abs}(X_{\mathcal{W}\text{sim}} - X_{\mathcal{W}\text{target}}) + \frac{\Sigma\epsilon_u + \Sigma\epsilon_\psi}{t_{sim}} \tag{8}$$

Here, $X_{\mathcal{W}\text{target}}$ represents the end of the environment, whereas $X_{\mathcal{W}\text{sim}}$ indicates the position of the MAV when the imposed flight properties are not satisfied. A low score $r_{penalty}$ indicates a good controller performance. The error signal ϵ represents the difference in the control set point and actual variable for u and heading. The previous term is divided by the duration of the simulation to stimulate controllers that last for a longer time in the environment. In other words, we give importance to a stable controller fulfilling the control task with less precision, than a controller with better control performance but more prone to violating the environment's contstraints.

Table 2. Sampling distribution of enviroment parameters

Parameter	Sampling Distribution
σ_u	$\mathcal{U}(0, 0.15)$ rad/s
σ_{u_P}	$\mathcal{U}(0, 0.25)$ rad/s
σ_γ	$\mathcal{U}(0, 0.15)$ rad
σ_{γ_P}	$\mathcal{U}(0, 0.25)$
δ_u	$\mathcal{U}(1, 4)$ steps

F. Simulation setup

For the environment, the 3D model from [19] is used in combination with the noise signals proposed in [20]. Similar to [19], a point-mass MAV is assumed that is under the influence of (ϕ, θ, ψ, T), where (ϕ, ψ, T) are controlled. As mentioned previously, θ is not controlled by the SNN controller, but set according to the variable β 2 if a landing location is acquired. Else, a different desired pitch setting θ can be chosen.

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}v_x\\ v_y\\ v_z\end{bmatrix},$$
$$\begin{bmatrix}\dot{v}_x\\ \dot{v}_y\\ \dot{v}_z\end{bmatrix} = \begin{bmatrix}0\\ 0\\ g\end{bmatrix} + \mathcal{R}_{\mathbf{C}}^{\mathbf{W}}\begin{bmatrix}0\\ 0\\ T\end{bmatrix} + \mathcal{R}_{\mathbf{C}}^{\mathbf{W}}\mathbf{K}\mathcal{R}_{\mathbf{W}}^{\mathbf{C}}\begin{bmatrix}v_x\\ v_y\\ v_z\end{bmatrix}, \tag{9}$$
$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\\ \dot{T}\end{bmatrix} = \begin{bmatrix}k_\phi(\phi^c - \phi)\\ k_\theta(\theta^c - \theta)\\ k_\psi(\psi^c - \psi)\\ k_T(T^c - T)\end{bmatrix}$$

Here, (x, y, z) is the position of the MAV in the World frame $\mathcal{W}$ and v_* is the velocity. $\mathbf{g}$ is the gravity effect and T the acceleration caused by thrust. Drag is estimated as a simplified first order drag matrix $\mathbf{K} = \text{diag}([-0.5, -0.5, 0])$. $\mathcal{R}_{\mathbf{C}}^{\mathbf{W}}$ is the rotation matrix from the World to Camera frame. The forces on the MAV caused by aerodynamics are given by $\mathcal{R}_{\mathbf{C}}^{\mathbf{W}}\mathbf{K}\mathcal{R}_{\mathbf{W}}^{\mathbf{C}}[v_x, v_y, v_z]^T$. A proportional feedback control system is in place for low level attitude control where the gains are $k_\phi = 6$, $k_\theta = 6$, $k_\psi = 6$ and $k_T = 6$. The model is a 10 states $\mathbf{x} = [x, y, z, v_x, v_y, v_z, \phi, \theta, \psi, T]^T$ and four input $\mathbf{u} = [\phi_c, \theta_c, \psi_c, T_c]^T$ nonlinear system. In order to increase real world transferability, the 3D simulation environment has artificial noise added to certain variables. These variables are calculated outside the control system in order to simulate real-time computation issues. The simulation is run at 200Hz. To increase robustness and adaptivity, the following noise signals are added to ventral flow u and heading error γ.

$$\hat{u}(t) = u(t - \delta_u \cdot \Delta t) + \mathcal{N}(0, \sigma_u^2) + u(t - \delta_u \cdot \Delta t) \cdot \mathcal{N}(0, \sigma_{u_{prop}}^2) \tag{10}$$

$$\gamma(t) = \gamma(t - \Delta t) + \mathcal{N}(0, \sigma_\gamma^2) \tag{11}$$

Furthermore, the observed ventral flow u is affected by σ_u and $\sigma_{u_{prop}}$, being the standard deviations of added noise and proportional noise, respectively. The noise parameters have been obtained by the work presented in [21]. The starting position in the environment is $(x_0 = 0, y_0 = 0, z_0 = 0.5)$ with speeds $(v_{x_0} = 0.4, v_{y_0} = 0, v_{z_0} = 0)$. The initial conditions do not match the set point u.

Table 3. Sampling distribution of network parameters

Parameter	Sampling Distribution
w_{ij}	$\mathcal{U}(w_{ij} - 0.05, w_{ij} + 0.05)$ clamped at [0.02, 1]
τ_{u_i}, τ_{X_i}	$\mathcal{U}(\tau - 0.02, \tau + 0.02)$ clamped at [0.3, 0.9]
θ_{u_i}	$\mathcal{U}(\theta_{u_i} - 0.02, \theta_{u_i} + 0.02)$ clamped at [0.6, 0.9]

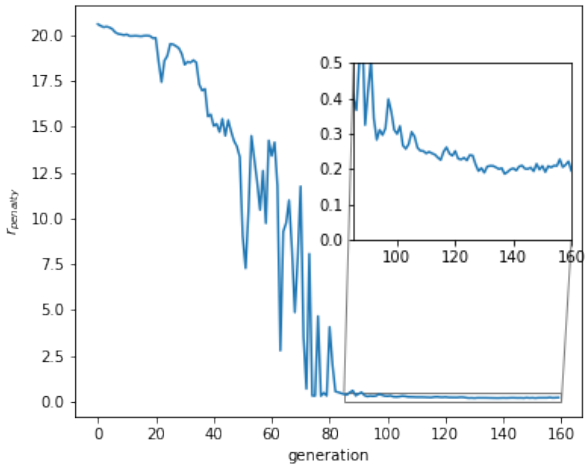


Figure 5. The learning curve shows objective evaluations of the best scoring individual every generation. The highlighted area shows that the controller has converged, which means that the MAV reaches the end of the simulation environment. When reaching the end, the only penalty given is due to a difference for u and ψ between the actual and setpoint

III. Simulation Experiments

50 individuals are initialized at the start of an evolution. The learning is stopped after the learning curve has converged ($\sigma_r < 0.02$ of last 25 generations). The best performing individual of the last 50 generations is considered for future simulations. The initial synaptic weights are drawn from $\mathcal{U}(0.1, 0.9)$, the thresholds from $\mathcal{U}(0.5, 0.9)$ and decay values from $\mathcal{U}(0.3, 0.9)$. In order to simulate SNNs, Pytorch in Python is used. The other network parameters are sampled from the distributions shown in table 3. NEAT-SNN evolves the network parameters, including topology. In contrast to other learned SNNs, only positive synapse weights are used. During evolution, the u flow set point is 1, which means that a larger error between actual and this set point results in a higher penalty (8).

The aim is learning a random topology SNN controller for a 3D flight control task. Maintaining a constant OF u rate allows the same controller to be used for take-off, landing and cruise. While the controller has only been learned on a cruise scenario, the line following controller uses the same learned flight strategy to perform take off and landings. In order to simulate the SNN in MLP fashion, the original topology has been transformed to fit a fixed architecture MLP. The number of hidden neurons differs from the actual hidden neurons in NEAT because of additional neurons needed to facilitate skip connections. While the network output remains the same, these connections that skip a layer require a placeholder neuron to transport a spike to the next layer. For comparison purposes, a conventional MLP network was supposed to be compared to a custom built sparse network. Unfortunately, a dense CNN would not converge, leading to an unusable controller. Therefore, only a single controller is presented. Figure 5 shows that the learning curve of the controller is not smooth. This is due to individuals suddenly being able to cover a larger part of the ground track. It is evident that at some point, an individual got 'lucky' and got past a certain corner of the sine wave, however this is no guarantee that this will happen again in the next generation. This explains the fallback in the objective evaluations scores $r_{penalty}$. After the 80th generation, the first part of the $r_{penalty}$ signal does not dictate the score any more and the time averaged u and ψ continue being optimized. This means that after 80 generations of 50 individuals being evaluated, the NEAT-SNN algorithm has been able to produce a sufficient SNN controller.

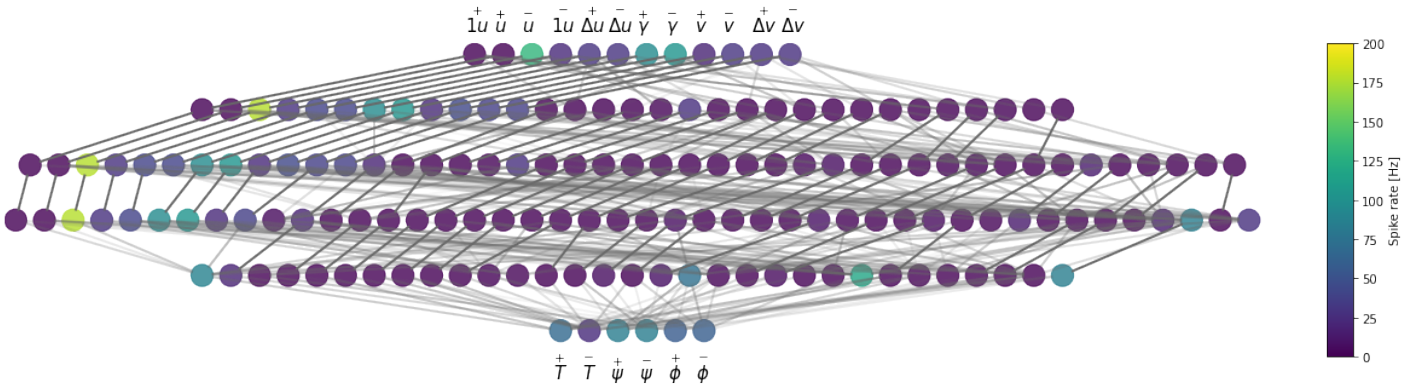


Figure 6. The transformed NEAT-SNN network fit for running on MLP architectures is shown. The thickness of the synapses represent the synapse weight, while the colours of neurons represent the spike rate. The neurons on the left, with parallel aligned synapses, are connections that span multiple layers. The lowest and last layer are the decoding neurons of which the colour depicts the average membrane potential, as these neurons do not fire spikes. The labels of the encoding and decoding neurons indicate the variables that the neuron represents. The labels including a 1, indicate that the neurons is fired when $|u| > 1$ depending on the sign of u.

The network in Figure 6 shows a transformed network of 167 neurons, while the same NEAT-SNN networks consists of 76 neurons. This denotes that an additional 89 neurons are needed to operate this network in a fixed MLP architecture. Of these 167 neurons, depending on the flight scenario, some do not spike. This either means that the inactive neurons were already present before learning started, or NEAT has mutated the parameters as such that the membrane potentials of these neurons never reach the threshold. An example of a network topology without neurons serving as synapses, can be found in 14 in the Appendix.

A. Take off

The take-off is initialized by increasing pitch, allowing the MAV to tilt forward and pick up horizontal speed U_C (Figure 7). Consequently, the u ventral flow will increase, whereafter the controller will increase thrust in order to gain altitude, keeping u constant. It is assumed that the ground surface allows some sliding for the MAV to speed up. Here, pitch is not gradually increased, but set to the desired angle immediately. The previous ensures the MAV reaches cruise phase as fast as possible.

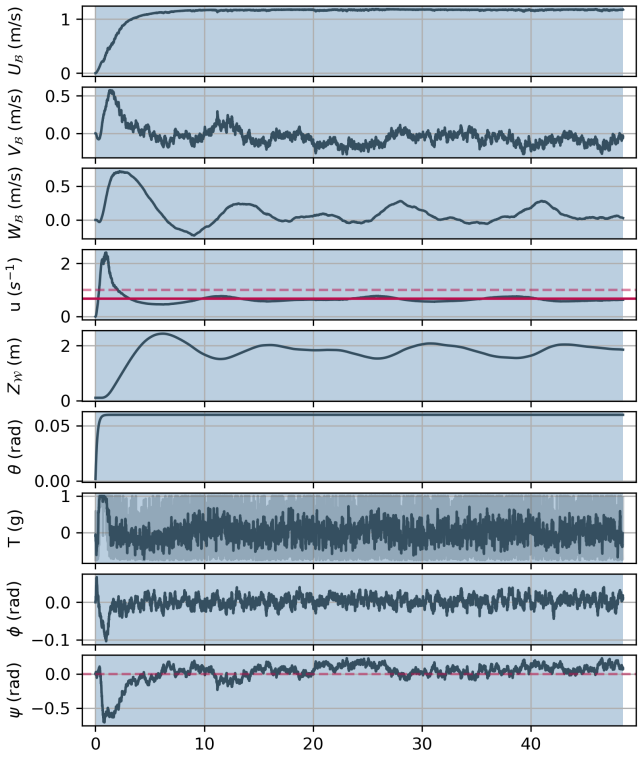


Figure 7. Body speeds U_c, V_c, W_c, ventral flow u, height, pitch, thrust, heading and roll for a simulated take-off. The dashed red lines indicate the set point of variables u and ψ during learning. The continuous red line shows the controller's actual set point for u. Since the thrust signal T is very noisy, an overlay is plotted that averages the last 10 time steps of T.

Figure 7 shows that the controller overshoots the equilibrium OF point by twice the equilibrium magnitude, but stabilizes after 5 s. While leaving the ground surface, we see that the controller is not maintaining the defined course from the start directly. A small error in heading is visible in the first 2–3 seconds of take off, which is corrected for after picking up some speed. Repeatedly executing this experiment shows that this is a negative ψ bias. This indicates that the deviation is not caused by noise, but by the controller itself. The trajectory in Figure 8 shows that no significant deviations from the ground path are present. One could argue that the controller is underdamped as some vertical swerving is noticeable after 30 s. This however, mainly is an issue at higher pitch angles. At lower pitch angles, vertical swerving is less evident.

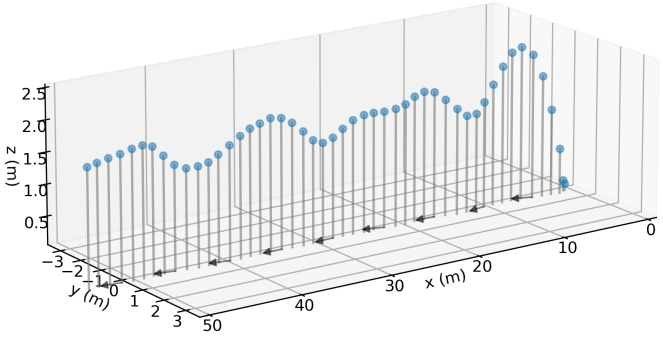


Figure 8. The dots indicate the position of the MAV, which is sampled every second. For more clarity, a bar is added that displays the position at ground surface level. The arrows indicate the direction of travel.

The thrust diagram in Figure 7 shows a broad range of calculated thrust commands. This is most likely due to the missing scaling factor between the decoding neuron and thrust control signal and negative thrust bias. This means that the SNN has learned taking into account the dynamics of the thrust control system of the environment. For future real world experiments, an additional controller will be needed. Since NEAT-SNN randomly assigns a pair of neurons with a new connection, the network possesses no symmetrical properties. One could expect that there would be a difference present in the flight commands following a set point error with a different sign. However, the roll ϕ and yaw ψ angle in the last two graphs of Figure 7, show no particular bias for either positive or negative set points. During take off, 84 out of 167 neurons in the network have not spiked. A large part of the network is inactive. However, as we will see during cruise, not all neurons remain unengaged. Some of these neurons are inactive due to moderate spiking activity of the neurons encoding the heading error. As the heading does not change over time, large errors are not recorded. The inactive neurons include neurons that function as a synapse. If a synapse is meant to bridge 2 layers and the first neuron remains inactive, the neurons after do not spike as well. The number of inactive neurons therefore seems higher than originally true without the fixed architecture framework.

B. Cruise

During cruise, the same ground path as the training scenario is followed. For demonstration purposes, a different pitch is given halfway in the experiment. The experiment starts at $\theta = 0.06$rad and at $x = 25m$, pitch is decreased to $\theta = 0.02$rad. The dots present in the trajectory graph, in the lower right corner of Figure 10, coincide with the pitch setting found in the lower left graph. These dots match the colour of shade in the graphs in Figure 9.

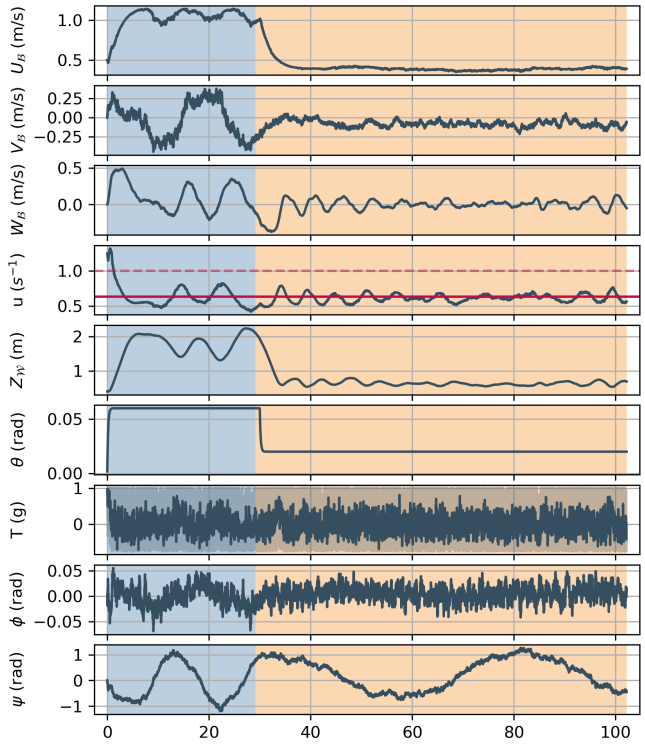


Figure 9. Body speeds U_c, V_c, W_c, ventral flow u, height, pitch, thrust, heading and roll for a simulated landing. The different coloured shaded areas indicate a different pitch setting and match the colours in Figure 10. Since the thrust output signal T is very noisy, an overlay is plotted that averages the last 10 time steps of T. The dashed red lines indicate the u set points during learning, as there is a discrepancy between the evolved set point and the actual u maintained.

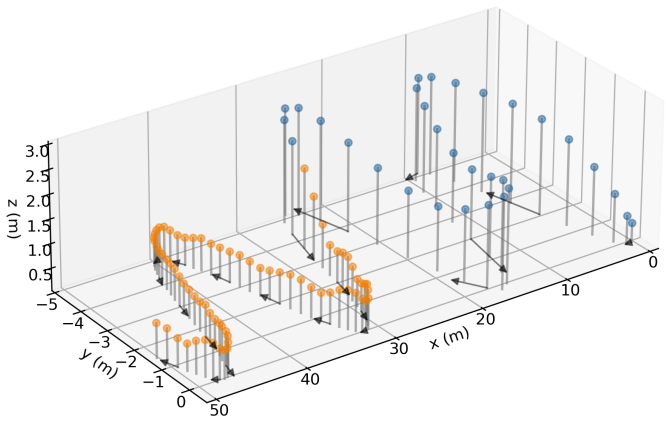


Figure 10. During cruise, a sine wave ground path is followed. The dots indicate the MAV's position and are sampled every second. Due to a lower pitch and consequently forward speed, the dots are closer to each other in the second part of the trajectory.

Although $u = 1$ is optimized for during learning, $u = 0.68$ is maintained in flight. The best performing controller seems to converge to such a value and does not improve further, as seen in the learning curve in Figure 5. This causes the learning curve to converge more closely towards $r_{penalty} = 0$. Interestingly, thrust is adjusted according to the pitch setting. This indicates that a single controller can find an equilibrium thrust level for different pitch settings. The continuous red line in the u graph in Figure 9 shows that the u set point remains the same, although pitch is changed. The SNN controller does not only calculate a simple thrust bias, as such a controller would only be useful for only a single pitch setting.

As explained in [13], a thrust bias is introduced that would correct for this lost vertical thrust force due to pitch. The controller in our work does not need a separate thrust bias calculation to be applied. Furthermore, the controller has learned to deploy roll and heading while making a turn. Although applied simultaneously, some drift in corners is visible as V_B increases (9). After corners, the MAV loses some altitude but recovers while approaching the next turn (10).

During cruise, 64 out of 167 neurons have not fired. This means that during turns, more neurons are engaged. Since for cruise, a sine wave ground path is followed, an increase in encoding spike activity has caused more neurons in the network to spike.

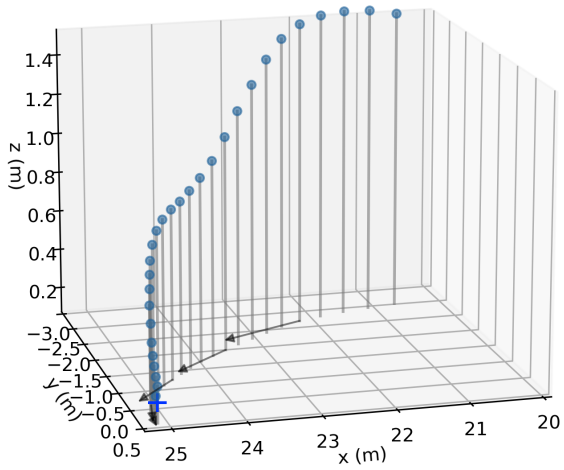


Figure 11. The dots indicate the position of the MAV, which is sampled every second. For more clarity, a bar is added that displays the position at ground surface level. A $0.25rad$ offset is introduced in the heading signal, which causes the MAV to approach the target with a slight curve.

C. Landing

The blue dots in Figure 11 represent the part of the trajectory that is shaded blue in the graphs of Figure 12. It is seen that until the very last moment before landing, a constant u is maintained. Comparable to take off, it is observable that at lower speeds, roll ϕ and yaw ψ angles experience more fluctuation. Instead of maintaining a certain pitch, like during take off and cruise, for landing, pitch is slowly decreased while approaching a target.

Again, β is the angle between the target and $X_C = 0$ line from the MAV point of view. Correspondingly, β will increase if the altitude decreases. Simultaneously, the closer the MAV gets to the target, the smaller β becomes. As a result, pitch will decrease, which in turn results in a lower altitude. This altitude decrease then increases β, but over time will cause β to decrease again as the MAV is nearing the target. This process repeats itself, which causes pitch θ to decrease in a 'wavy' manner, as shown in Figure 12. Pitch will roughly be equal to zero near the target. The horizontal speed the MAV is still carrying while touching down, is not caused by the pitch setting at that moment, but by the remainder speed of the approach to the target.

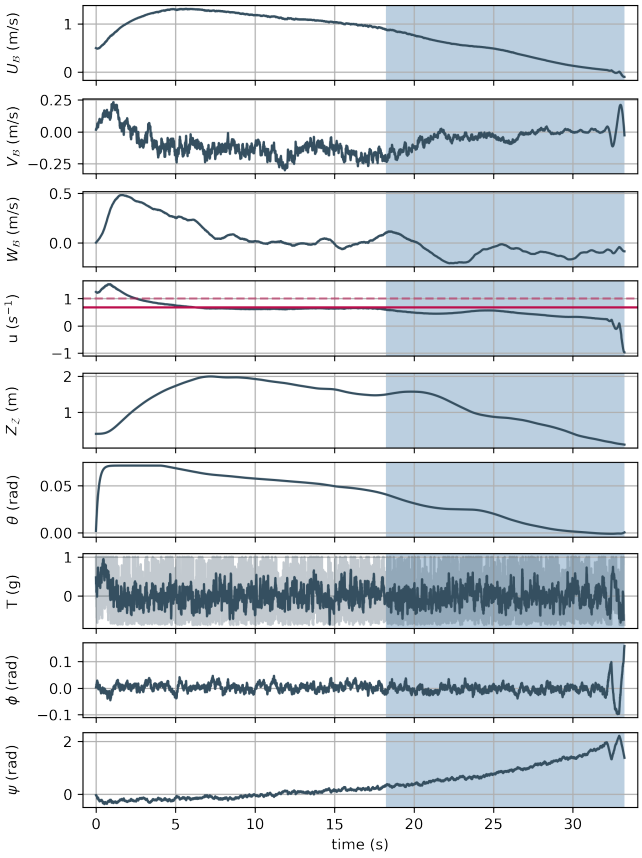


Figure 12. Body speeds U_c, V_c, W_c, ventral flow u, height, pitch, thrust, heading and roll for a simulated landing. The blue shaded area on the right half of the graphs define from where the trajectory becomes visible in Figure 11. Since the thrust output signal T is very noisy, an overlay is plotted that averages the last 10 time steps of T. The dashed red lines indicate the u set points during learning. As there is a discrepancy between the evolved set point and the actual u maintained, the red line (experimentally maintained u) highlights the difference.

Divergence based landings are prone to self-induced oscillations [22]. These oscillations are a result of scaled uncertainties in OF control while nearing the ground surface. While some moderate oscillations are observed, altitude gains are not visible in Figure 11. The proposed landing strategy introduces a small off set in ψ, which causes the MAV to approach the target with a slight curvature. 100 landing are executed, of which the touch-down position error are shown in Figure 13. The box plot tells us the mean distance from the target is a couple of centimetres at touch-down. Whilst the controller does not match the performance in terms of speed and precision with conventional control strategies (Kalman filter, PID, MPC), the soft landings and overall precision ensure reproducibility with a lower level of algorithmic and sensory intelligence.

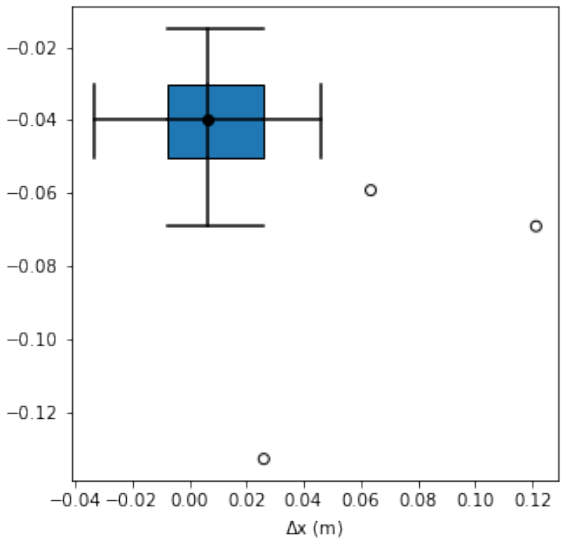


Figure 13. A 2D box plot of the touch-down locations of 100 executed landings.

IV. Conclusion

It is shown that a single controller can be evolved for full 3D neuromorphic control. Making use of a challenging learning environment, a single controller can be utilized for take-off, cruise and landing. The learning scheme is less focused on conventional control performance metrics but instead, puts many constraints on the flight path of the MAV. This will decrease the probability of reaching unstable states in flight. This results in a robust and adaptive controller that can operate in different circumstances. Though, these flight operation prerequisites must be determined before learning and influence the feasibility of the proposed learning metrics. While a $u = 1$ set point was chosen during evolution, a clearly different set point was accomplished.

References

[1] Balamurugan, G., Valarmathi, J., and Naidu, V. P. S., "Survey on UAV navigation in GPS denied environments," *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, IEEE, Paralakhemundi, Odisha, India, 2016, pp. 198–204. doi: 10.1109/SCOPES.2016.7955787, URL `http://ieeexplore.ieee.org/document/7955787/`.

[2] Yu, W., Sanchez, E. N., and Kacprzyk, J. (eds.), *Advances in Computational Intelligence*, Advances in Intelligent and Soft Computing, Vol. 116, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi: 10.1007/978-3-642-03156-4, URL http://link.springer.com/10.1007/978-3-642-03156-4.

[3] Neftci, E. O., Mostafa, H., and Zenke, F., "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks," *IEEE Signal Processing Magazine*, Vol. 36, No. 6, 2019, pp. 51–63. doi: 10.1109/MSP.2019.2931595, URL https://ieeexplore.ieee.org/document/8891809/.

[4] O'Shea, K., and Nash, R., "An Introduction to Convolutional Neural Networks," , Dec. 2015. URL http://arxiv.org/abs/1511.08458, arXiv:1511.08458 [cs].

[5] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A., "Automatic differentiation in PyTorch," 2019, pp. 8024–8035.

[6] Stagsted, R., Vitale, A., Binz, J., Renner, A., Bonde Larsen, L., and Sandamirskaya, Y., "Towards neuromorphic control: A spiking neural network based PID controller for UAV," *Robotics: Science and Systems XVI*, Robotics: Science and Systems Foundation, 2020. doi: 10.15607/RSS.2020.XVI.074, URL http://www.roboticsproceedings.org/rss16/p074.pdf.

[7] Hagenaars, J. J., Paredes-Valles, F., Bohte, S. M., and de Croon, G. C. H. E., "Evolved Neuromorphic Control for High Speed Divergence-Based Landings of MAVs," *IEEE Robotics and Automation Letters*, Vol. 5, No. 4, 2020, pp. 6239–6246. doi: 10.1109/LRA.2020.3012129, URL https://ieeexplore.ieee.org/document/9149674/.

[8] Stroobants, S., Dupeyroux, J., and de Croon, G. C. H. E., "Neuromorphic computing for attitude estimation onboard quadrotors," *Neuromorphic Computing and Engineering*, Vol. 2, No. 3, 2022, p. 034005. doi: 10.1088/2634-4386/ac7ee0, URL https://iopscience.iop.org/article/10.1088/2634-4386/ac7ee0.

[9] Brandli, C., Berner, R., Minhao Yang, Shih-Chii Liu, and Delbruck, T., "A 240 × 180 130 dB 3 µs Latency Global Shutter Spatiotemporal Vision Sensor," *IEEE Journal of Solid-State Circuits*, Vol. 49, No. 10, 2014, pp. 2333–2341. doi: 10.1109/JSSC.2014.2342715, URL https://ieeexplore.ieee.org/document/6889103.

[10] de Croon, G. C. H. E., De Wagter, C., and Seidl, T., "Enhancing optical-flow-based control by learning visual appearance cues for flying robots," *Nature Machine Intelligence*, Vol. 3, No. 1, 2021, pp. 33–41. doi: 10.1038/s42256-020-00279-7, URL http://www.nature.com/articles/s42256-020-00279-7.

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

[12] Ho, H., De Wagter, C., Remes, B., and de Croon, G., "Optical-flow based self-supervised learning of obstacle appearance applied to MAV landing," *Robotics and Autonomous Systems*, Vol. 100, 2018, pp. 78–94. doi: 10.1016/j.robot.2017.10.004, URL https://linkinghub.elsevier.com/retrieve/pii/S0921889017305626.

[13] Ruffier, F., and Franceschini, N., "Optic flow regulation: the key to aircraft automatic guidance," *Robotics and Autonomous Systems*, Vol. 50, No. 4, 2005, pp. 177–194. doi: 10.1016/j.robot.2004.09.016, URL https://linkinghub.elsevier.com/retrieve/pii/S0921889004001733.

[14] Stanley, K., and Miikkulainen, R., "Efficient evolution of neural network topologies," *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, Vol. 2, IEEE, Honolulu, HI, USA, 2002, pp. 1757–1762. doi: 10.1109/CEC.2002.1004508, URL http://ieeexplore.ieee.org/document/1004508/.

[15] Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.-K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y.-H., Wild, A., Yang, Y., and Wang, H., "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, Vol. 38, No. 1, 2018, pp. 82–99. doi: 10.1109/MM.2018.112130359, URL http://ieeexplore.ieee.org/document/8259423/.

[16] Khan, M., Lester, D., Plana, L., Rast, A., Jin, X., Painkras, E., and Furber, S., "SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor," *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, IEEE, Hong Kong, China, 2008, pp. 2849–2856. doi: 10.1109/IJCNN.2008.4634199, URL http://ieeexplore.ieee.org/document/4634199/.

[17] Hagberg, A., and Swart, P., "Exploring network structure, dynamics, and function using NetworkX," , 2008. URL https://www.osti.gov/biblio/960616.

[18] Perlin, K., "Improving noise," *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM, San Antonio Texas, 2002, pp. 681–682. doi: 10.1145/566570.566636, URL https://dl.acm.org/doi/10.1145/566570.566636.

[19] Li, S., Horst, E., Duernay, P., De Wagter, C., and Croon, G. C. H. E., "Visual model-predictive localization for computationally efficient autonomous racing of a 72-g drone," *Journal of Field Robotics*, Vol. 37, No. 4, 2020, pp. 667–692. doi: 10.1002/rob.21956, URL https://onlinelibrary.wiley.com/doi/10.1002/rob.21956.

[20] Scheper, K. Y., and de Croon, G. C., "Evolution of robust high speed optical-flow-based landing for autonomous MAVs," *Robotics and Autonomous Systems*, Vol. 124, 2020, p. 103380. doi: 10.1016/j.robot.2019.103380, URL https://linkinghub.elsevier.com/retrieve/pii/S0921889019302404.

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

[22] Howard, D., and Kendoul, F., "Towards Evolved Time to Contact Neurocontrollers for Quadcopters," *Artificial Life and Computational Intelligence*, Vol. 9592, edited by T. Ray, R. Sarker, and X. Li, Springer International Publishing, Cham, 2016, pp. 336–347. doi: 10.1007/978-3-319-28270-$1_2$8, *URL*, series Title: Lecture Notes in Computer Science.

Appendix

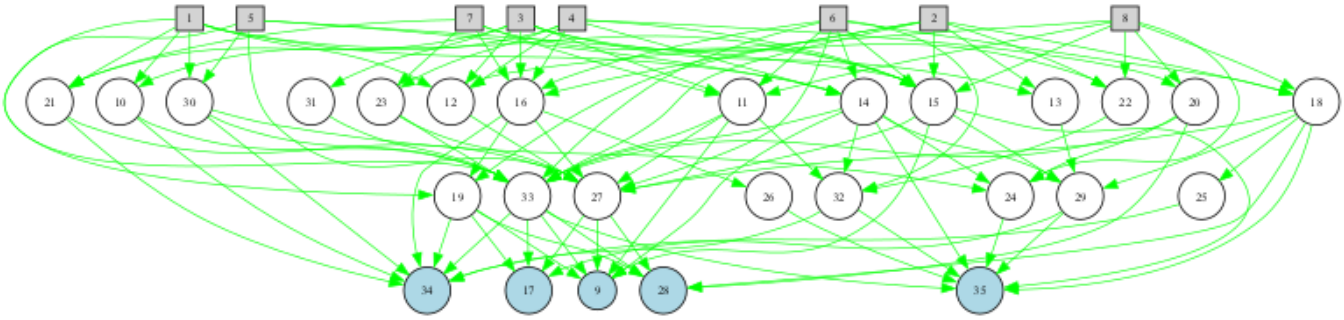


Figure 14. An evolved network using the NEAT-SNN module. The square nodes in the top layer represent the input neurons while the blue shaded neurons on the bottom are the output neurons. The remaining neurons are part of the hidden layers. It should be noted that the networks starts learning without any direction connection between the input and output layer. This means that every input signal travels through a neuron before reaching the output neurons.

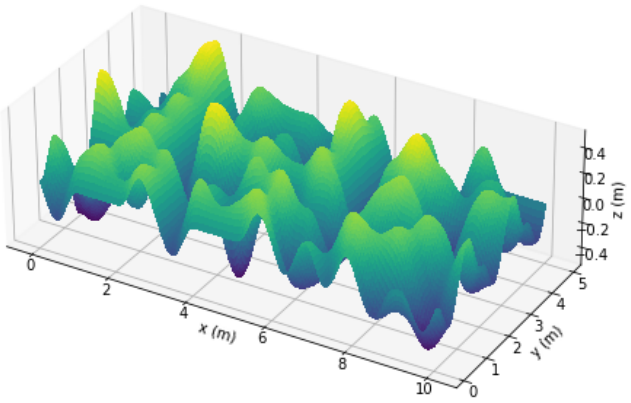


Figure 15. Perlin noise adds a realistic terrain profile to a ground surface. By training on such ground surface, a robuster controller is evolved.

Part II

Literature Study

4

State-of-the-art Landing controllers MAV's

Since the appearance of MAV's, an increasing interest towards automation occurred in parallel. Not only has effort been put in automating the procedures and execution of mission specific tasks when deployed on a MAV platform, but also have fully autonomous flight control solutions, that are both scalable and resource efficient, been searched for. Flight control solutions can be categorized by their level of autonomy and flight phase. In general, landing is regarded as the most critical and challenging flight phase because of increased exposure to risks and reliability issues induced by for example changing conditions. Exemplary properties that can be tracked to demonstrate the performance over other control solutions are resilience to disturbances and noise or landing speed in general. Before researching the possibly advantageous performance gains of neuromorphic control for Unmanned Aerial Vehicles (UAVs), more consideration should be solicited to the current landscape of (landing) controllers available for UAV's. The following section will briefly analyse while categorizing the present controllers researched and experimented with.

In order for an (UAV) to complete an autonomous navigational task, such as landing, two main aspects must be considered, namely sensing and control. Although many sensing and control methods may dictate one another method in terms of applicability and possible configurations, some are compatible with other either sensing and control techniques and therefore are initially discussed separately.

4.1. Sensing

Three main sensing techniques are identified: Global Positioning (GPS), Inertial Navigation Sensors (INS), vision-based sensing as mentioned in [7]. For completeness purposes, all options are briefly analysed while examining their technological applicability for the research goal.

1. **GPS.** Landing techniques that are based on GPS use a signal to estimate the position of a drone. The signal(s) received by a MAV are transmitted by for instance satellites or other comparable external communication infrastructures. In order for a MAV to determine its exact position, multiple signals are required. Additionally, the precision of location synchronization generally increases if the number of incoming signals grows. The advantages of a GPS based landing technique are the potential precise position estimates and since GPS technology is on of the oldest navigational systems used, many algorithms and experience exist in using such intelligence. Unfortunately, GPS can be inaccurate when the drone is not 'visible' and the processing power for calculations on the receiving end, make GPS expensive. Furthermore, GPS does not allow local perceptibility, meaning no information regarding the actual environment is given. GPS however is resistant to different atmospheric conditions, but it's efficacy for indoor use is compromised by disturbances originating from constructions and buildings.
2. **INS.** The sensors inhibit some mechanical component that tracks a certain physical exertion and links the measurement with accelerations. On the upside, INS are not dependent on external infrastructure, whereby the UAV is fully contingent on its own efficacy. Since INS involves some physical component, this often induces sensor drift. When drift is experienced, the sensors are influenced by a low frequency change over time, which often does not directly affect high frequency usage. However, over longer periods of times, an off set might be introduced. Drift can often be corrected for when continuous and invariable changes are experienced, but need a different sensory

suite or system for calibration. Another aspect to consider is the excluding additional environmental awareness since only intro observable states are utilized.

3. **Vision-based.** Vision based navigational features can be divided into two categories, namely solely operating visionary guidance systems and computer vision algorithms that are part of the feedback control loop. The first category includes systems that are capable of detection and tracking, however, can not be used as a stand alone autonomous algorithm. Instead, these supporting systems can be combined with other navigational techniques to increase overall performance or expand navigational capabilities. The second group addresses flight control systems that are fully dependent on visionary-based observables. Additionally, camera and processors are light and tiny, which improves scalability and costs.

4.2. Conventional control algorithms

The following section will introduce different kind of MAV controllers. The controllers are categorized by methodology, which in turn can have multiple versions. The final section introduces intelligent control using different kinds of Machine Learning (ML) techniques

4.2.1. Proportional-Integrative-Derivative (Linear)

A Proportional-Integrative-Derivative (PID) controller is one of the most common feedback control techniques. The controller continuously calculates the control input signal by tracking the error between a set point value and the actual process variable. By applying a certain gain multiplication to the proportional K_p, integral K_I and derivative term K_D of the signal to calculate control signal u, a control loop mechanism is created 4.1.

$$u = K_p e + K_I \int_0^t e dt + K_D \dot{e} \tag{4.1}$$

To this day, PID control still remains of the most widely just control technology due to its simple and robust set up as shown in [8]. Although less complex control structures and configuration are achievable, at 91%, the PID controller is regarded to be the most influential control technology to date [8]. As previously mentioned, a PID controller is easy to implement and tune, however, complex control solutions are limited. Although tuning is relatively easy, an accurate model representing the actual behaviour of a system is necessary. The latter results in the control mechanism not having any adaptive properties. The predetermined gains dictate the behaviour of a system. In order to change performance parameters such as settling time and overshoot, the PID gains must be tuned again. As shown in [9], a PID controller can be used as a navigational control mechanism by controlling the states of a quad rotor. Since the system is under-actuated, the flight angles are controlled for. In [10] the authors as well use a PID controller, however do calculate the desired speeds for the earth-fixed frame, showing the versatility of a PID controller.

4.2.2. Optimization based control

Optimized control design makes use of knowing how valuable it is to be in a certain state, especially with respect to other possible states. Two main optimal control methods are categorized, namely methods based on cost functions and methods that calculate the optimal control input by simulating possible control inputs while establishing the corresponding reward. Both methods need to have some value granted to the importance of certain states and/or control input (combinations) in order to express a preference. While some might refer to cost and others prefer reward, the same is implied. The next two subsections will discuss the advancements in Linear Quadratic Regulators (LQR) and Model Predictive Control (MPC) based control.

LQR

A LQR control mechanism favours the control of a certain state over other states, and defines how the use of different control inputs should be prioritized or minimized over other inputs. Equation 4.2.2 displays the adjustable matrices Q and R that dictate the shape of the cost function J. The cost function is minimized by the LQR controller in order to calculate the optimal control strategy considering states x and control input u.

$$J = \int_0^\infty (x^T Q x + u^T R u) dt$$

By solving the equation, a feedback gain K can be calculated. Large values in the Q matrix will result in a larger gain, ensuring a faster response. Though, a larger R value will penalize the use of a control input and decrease the feedback gain value.

While many applications only consider systems with certain physical continuity, other applications demonstrate the LQR capabilities of operating under severe physical modifications. The authors in [11] demonstrate the robustness of a LQR controller by showing the stability properties attained by a quadcopter despite having lost multiple rotors using LQR. As mentioned before, a LQR controller can penalize the use of control inputs, which can prove beneficial for changing flight conditions. By severely punishing the use of, in this case a certain rotor, eliminating the general control possibility, a new stable equilibrium can be found. Adjustments in the to be optimized cost function, allow the drone to still be controlled for position. However, in this case, attitude control is sacrificed since full system controllability is not possible any more. Not only do vehicles that are damaged benefit from such control properties, LQR's are also beneficial for platforms that can change their configuration in-flight. This allows the system to achieve better flight control qualities while maintaining the same control system [12]. Changed flight characteristics due to morphology, which signifies modifications to the shape of the vehicle, can be corrected for using an LQR controller. A disadvantage of these systems is that a mechanism must be in place that actively monitors the vehicle's state. This mechanism will initiate a computation for new feedback gains in the case of control preferences.

LQG Another form of a LQR controller is the Linear Quadratic Gaussian (LQG) controller. A LQG consists out of a LQR and a Linear Quadratic Estimator (LQE), which is a Kalman Filter, that experiences Gaussian distributed disturbances. A convenient property of the LQG is, when multiple cost functions co-exist, an optimal control strategy can still be found. It should be noted though that there remain specific circumstances in where instabilities might occur. Unmodelled non-linear dynamics and system uncertainties might lead to unstable situations. The authors in [13] make use of the additional state estimation and filtering capabilities. Here, a LQG is favoured over a LQR, since the discontinuities introduced by the SLAM algorithm, migh introduce small offsets and noise in the control input.

Model Predictive Control

A Model Predictive Control (MPC) system essentially performs multiple actuator deflection (i.e. control input) strategies and forecasts future states using a system model. By simulating possible control inputs (random or planned) over a certain amount of time steps in the future, a score is assigned to every trajectory. The mechanism performs an optimisation using constraints and a cost function. As the highest scoring sequence of control inputs is known, the first step of the input sequence is employed as the next control input. This sequence of computations is performed for every time step, meaning every $t + 1$ time window is optimized for. When projected in the form of a landing controller, [14] demonstrates the efficacy of MPC with a simple and straightforward cost function for a landing requirement. By selecting the control strategy that transports the drone as close as possible to the moving landing target, a successful landing is accomplished.

Traditional MPC algorithms mainly concern the planning aspect, namely by assessing possible combinations of future states and input combinations. The planning mechanism is responsible for producing different sequences of inputs. Since planning consists of many aspects, much efficiency and performance is to be gained by advancing certain elements of the MPC algorithm. Several MPC adaptations exist, where computation speed and precision are significantly improved by optimizing trajectory generation and tracking.

As for most MPC algorithms, a linear(ized) model is used for the sake of modelling and computation speeds. A natural consequence is the gap between simulation and reality. These model uncertainties can consist of non-linear aerodynamic effects and payload variations and decrease the overall system performance. Non-Linear MPC (NMPC) adjusts for many non-linearities. The authors in [15] capture these model uncertainties and compensate these by mapping the differences between estimated and reference future states. The adaptive controller uses a reference model that learns the discovered and matched uncertainties to compensate for immediately. It must be noted that model imperfections are, because of the many iterations, minimized over time. Since the modelling error stays consistent over time, optimal sequences of control inputs can still be produced [16]. The same accounts for [17] where a high speed and aggressive control task is carried out. A radio-controlled car is raced around a track. Although the model is

not the best representation of true environment dynamics, high speeds are accomplished. Although not a flight control example, this shows that, while a poor model is being used, appropriate control commands can still be calculated

4.2.3. Non-linear

The difference between linear control and non-linear control is that non-linear control does not conform with the superposition principle. Additionally, the stability of non-linear systems heavily depend on initial conditions, input signal properties and internal system dynamics. Control methods exist that omit these non-linear terms as much as possible. The techniques that will be discussed are Sliding Mode control and Incremental Non-linear Dynamic Inversion.

Sliding Mode Control

Sliding Mode Control (SMC) is a robust non-linear control method that applies a discontinuous control signal to a non-linear system. Due to its insensitivity to disturbances and uncertainties, the method provides a robust approach for controlling such systems. The idea behind SMC is to drive the controlled states onto a particular surface within the state space. The latter is referred to as sliding surface and must be chosen for carefully. Part of the design of a SMC thus involves choosing appropriate states, that will allow the controller to keep the states close to the sliding surface for convergence. Depending on the initial conditions of a system, the controller will drive the states to the sliding mode surface, which is referred to as reaching mode. After the states have reached the surface within the state space, sliding mode is initiated and the system will 'steer' near to the surface as close as possible. This control rule can battle non-linear effects, however, outputs a harsh discontinuous control signal that might induce chattering. The oscillations caused by these high frequency control signals are generally less desired. Although several adaptations are known, these will not be discussed as this is outside the scope. Adaptations include smoothing functions etc.

As mentioned previously, SMC can be used in combination with another feedback control mechanism, as done in [18]. The authors successfully implemented a sliding mode controller that made the controller adaptive by letting a sliding surface function dictate the PID gains. Overall, the controller outperformed an ordinary PID controller but only if the learning rates, that are responsible for adjusting the PID gains, are large enough.

Incremental Non-linear Dynamic Inversion

Non-linear Dynamic Inversion (NDI) is a model-based control system that describes a linear relationship between input and output. The control input is calculated by inverting a physical model. Incremental NDI (INDI) is less dependent on model parameters, but rather a sensor-based control approach. A disadvantage of NDI is that the control system must know the full system. An accurate model is expensive and hard to obtain. As INDI is more dependent on sensor readings, it is important these are adequately filtered.

The authors in [19] achieves robust and adaptive attitude control for a MAV using INDI. Additionally, some measure of control effectiveness is expressed, since the system is heavily dependent on inverting the model of the controls. The aerodynamic uncertainties do not need any modelling, since the effects that they have on the system, are measured with angular acceleration. This would allow basic morphology without any tuning of system parameters. An optical flow control application is combined with Extended INDI (EINDI) in [20]. Constant divergence landings are performed that can be varied for height while not experiencing oscillations [21]. Most important contribution is the lacking dependency of having a set timescale for the input-output model, which allows such controller to be used for scaleless properties.

4.3. Intelligent control methods

Intelligent control methods are confined into a separate section, as this category includes the usage of artificial intelligence to learn the most optimal control behaviour (policy). Since neuromorphic control is also regarded as artificial intelligence, the learning (exploration and exploitation), deployment and many other practical application complications overlap with the same complications as experienced for intelligent control methods. Therefore, more emphasis is put on the obstacles and results found for designing flight (landing) controllers with machine learning in a separate section.

Intelligent control methods belong to a category of methods that use various Artificial Intelligence (AI) computing techniques, such as Neural Networks (NN) and Machine Learning (ML). A NN is a decision-

making framework that inherits bio inspired properties that can be deployed to construct a control policy. The most optimal decision variables can be found by ML techniques that either train or evolve a NN. Intelligent control systems learn a policy, with or without a model, to control a dynamical system. The policy should theoretically perform better each iteration, since it is learning to execute (control) a certain task and has more attempts as the number of possible interactions with the environment increase. Fuzzy control is also regarded as an intelligent control method, since fuzzy corresponds to a certain logic that the designer can understand. This 'logic' can address non-linearities or other uncertainties that hard to coop with for other traditional controllers.

AI includes a broad range of techniques and algorithms that can be deployed for control tasks in numerous ways. An apparent distinction can be made between techniques dealing with low level motor commands and higher level control tasks. These higher level control tasks are supported by NN algorithms that compute features as waypoints, speeds and trajectories but are not involved in directly calculating the appropriate motor commands for control. The NN based algorithms supporting or optimizing high level control tasks are hybrids since the control system necessitates another controller like the controllers mentioned in this chapter. [22] is one of the first to introduce NN's for computing motor commands based on previous states and control input. Two NNs are utilized that support the determination of uncertainties and non-linearities to perform a non-linear inversion. On the other hand, [23] for instance, has the AI part compute optimal states for a MAV to be in at certain locations. Whilst learning high-level policies, a secondary low level control systems calculates the actual motor commands. In [24], the authors use MPC to calculate the actual motor commands.

Due to the bio inspired computational properties of NNs, non-linearities can be captured well [25]. Generally, for classic control applications, it is desirable to have a linearized system. This unfortunately leaves a part of the dynamics and model uncertainties to be unmodelled, which is undesirable. NN's in control systems either estimate or compensate for uncertainties or provide full state feedback control inputs. The first category includes many examples that implement NNs for reducing the model error. Not only can this be done by creating a partial model that can approximate the non-linear effects, but also by learning a network to perform model inversion for calculating the appropriate control commands. Different implementations exist considering reducing the model error. [26] calculates the uncertain disturbance force that should be added to the control input in order to follow a landing trajectory. The training is performed offline. The other category directly calculates the appropriate control input. The authors in [27] use a NN feedback controller which is trained online by trying to recreate control input from the linear and non-linear feedback controller. [28] utilizes a simple control law that guarantees convergence, although this experiment is not performed in a real-world test. It should be noted that many applications are not performed in real-world tests, but rather simulations.

5

Vision-based landings

Positioning systems that are dependent on signals from external infrastructures are often expensive, sensitive to reflective surfaces and interference in, especially in-door environments. Cameras are small, cheap and work in a large variety of environmental conditions while possessing a high degree of employability. Additionally, biological system have proven that visual information can be utilized (sometimes in combination with other senses) for safe determination of ego-motion and subsequently navigation. The following chapter will discuss methods to extract, model and use visual features for landing.

5.1. Optical Flow

Optical flow (OF) is a visual cue that expresses the difference due to motion between two sequential frames. This difference signifies a direction and magnitude and helps to determine ego-motion, obstacle detection and can be a navigational aid. OF cues provide smaller animals like birds and insects the ability to quickly avoid obstacles and navigate through narrow spaces. For instance, for landings, insects mainly depend on the divergence rate, which is kept constant during landing [29]. Additionally, OF is used to determine the travelled distance using the Focus of Expansion (FoE) [30] and to regulate speed of flight [31]. Insects use observed shifts in brightness patterns, that are further processed to eventual visual features, for complex analysis of their image in order to identify shapes and motion. In a sense, insects set a benchmark for autonomous MAV's in terms of performance vs. efficiency of understanding and acting upon their environment, considering their size. Determining OF through computer vision necessitates power and hardware, which is scarce on a MAV platform. Many algorithms with different computational approaches exist that can estimate OF [32]. The following section will show different OF estimation methods and after discuss the modelling methods, including the constraints and assumptions that lead to different visual observables.

5.1.1. Estimation

As discussed before, OF refers to the displacement of intensity patterns over time on the camera's sensor. OF can be determined using conventional frame-based camera's, event-based sensors and OF sensors. The analysis and processing of OF through event-based vision will be treated in the last section. Apart from OF sensors, most frame-based OF algorithms use the brightness constancy assumption as mentioned in [33]. The assumption states that intensity I of a local region of the viewed image plane remains equal over a short time period 5.1:

$$\frac{\delta I}{\delta x}\frac{\delta x}{\delta t} + \frac{\delta I}{\delta y}\frac{\delta y}{\delta t} + \frac{\delta I}{\delta t} = 0 \tag{5.1}$$

Here x and y represent the pixel position on the sensor while $\frac{dx}{dt}$ and $\frac{dy}{dt}$ are the OF component of u and v respectively. At least two consecutive frames are needed to obtain partial derivatives $\frac{dI}{dx}$ and $\frac{dI}{dx}$. Although not as important as the brightness constraint, one must adhere to a fast enough sampling frequency and adequate spatial smoothness. The sampling frequency of the video must be fast enough to capture motion. Furthermore, an adequate number of pixels must be present that can sufficiently differentiate motions.

Considering the 2D image plane when having only one brightness consistency constraint, two unknown components u and v must be calculated. The latter described under-constrained condition is also known

as the aperture problem. For instance, since strictly horizontal or vertical edges only allow for normal flow extraction (one dimensional motion), optical flow can not be estimated. Normal flow is the motion detected normal to the contour's direction. These contours are lacking distinguishable corner points, which is why only normal flow is estimated.

In order to determine the exact u and v components of OF, another constraint is required. Multiple OF estimation algorithms exist that can be classified according to the different kind of secondary assumption made.

Gradient-based estimation methods assume that the observed flows u and v are constant among neighbouring pixels. Local and global gradient-based methods exist. The authors in [33] created one of the first OF algorithms that included every pixel in the optimization process. By means of minimization, a least squares solution for the best fitting OF is calculated. An additional constraint has later been added to decrease computational loads. The smoothness constraint introduced an iteratively corrected factor that would assume no discontinuities in consecutive frames of the observed flow. This is regarded as a global OF optimization method, since every point in the image plane is used for calculation.

A local approach only optimizes for a certain number of features/points on the image plane. Sparse methods benefit from choosing features as corners, since more brightness variance between neighbouring pixels is present. These regions with divergent pixel values allow for more precise displacement estimates to be produced. Before comparing the same visual feature on two different images and calculating the subsequent optical flow, one must determine which features to track. In [34], the authors use corners and edges for the Features from Accelerated Segment Test (FAST) algorithm. The quality of features tracked influences the accuracy of the OF calculation. Using feature monitoring as done in [35], optimal OF estimation is pursued. The most popular, though one of the oldest, OF estimation algorithm is the Lukas-Kanade estimation method. Here, [36] assumes that the flow is constant in small neighbourhoods and subsequently minimizes the witnessed brightness shift within a small neighbourhood.

Determining OF through correlation methods, means localizing a certain pixel patch on a successive frame to determine its motion. By simulating the motion of a pixel patch for different possible displacements, a subsequent matching motion can be found. A matching function minimizes the difference over a certain search area, which is a squared window patch, as shown in [37]. A general matching function that minimizes the sum of squared difference can be written as:

$$\underset{dx,dy}{\arg\min} \Sigma(I_t(x,y) - I_{t+dt}(x+dx, y+dy))^2 \tag{5.2}$$

Here I is the intensity of a pixel and is compared with another pixel in the neighbourhood on a frame at $t+1$. Since optical flow vectors represent a velocity of a certain distance over time, block matches can be found by varying the spatial or temporal component. An advantage of temporal searching methods is that complexity increases linearly with the number of frames, while the complexity for spatial methods increases quadratically with the size of the window.

5.2. Modelling

The following section will define an OF model that is used to link observed flows to ego-motion. For future simplifications and derivations, the following formulated relationships will serve as a base. In order to start interpreting viewed motion, the OF parameters must be determined.

The model is based on [38], which assumes that the retina or sensor can be viewed as a plane and the camera's aperture is viewed as a pinhole-point. The position of a point in both the world frame (W) and camera frame (C) is defined by (X_w, Y_w, Z_w) and, (X_c, Y_c, Z_c) respectively. (U_c, V_c, W_c) are the corresponding velocity components of a point in the (C) frame. The Euler angles (ϕ, θ, ψ) express the orientation of frame (C) with respect to (W) and represent roll, pitch and yaw respectively. Additionally, (p, q, r) are the corresponding rotational rates. Furthermore, the camera pixel coordinates are (x, y) and their velocities, which represent OF components, denote (u, v). The exact labels and axis can be found in figure 5.1.

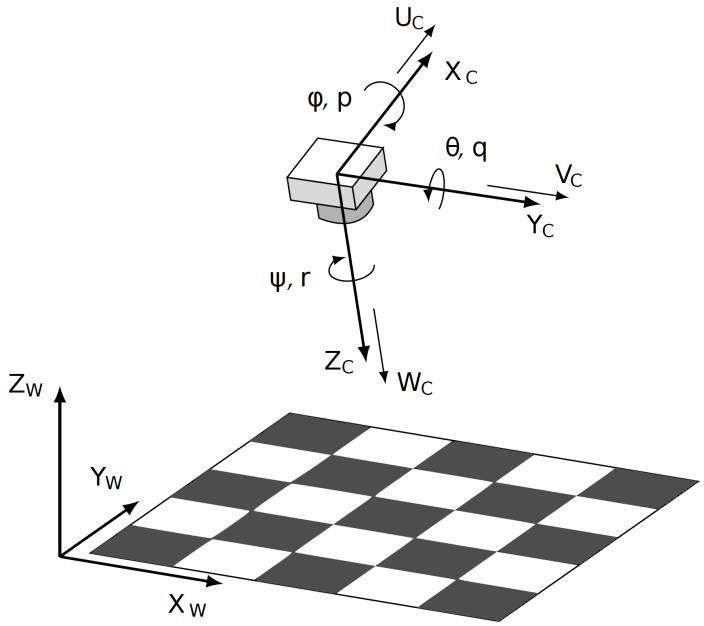


Figure 5.1: Adopted from [39]. Two reference frames are present, namely of the world W and camera C. The motion of the camera C is described by the translational velocities (U, V, W), the Euler angles (ϕ, θ, ψ) and rotational rates (p, q, r).

The projection of an arbitrary point in the reference frame (C) onto the image plane is denoted by $(x, y) = (\frac{X_C}{Z_C}, \frac{Y_C}{Z_C})$. Due to the observer's motion, this point moves with velocity (u, v) across the image plane. Utilizing the time derivative of points on the image plane, the OF of a point can be expressed into translational and rotational components 5.3, 5.4.

$$u = -\frac{U_c}{Z_c} + U_c\frac{W_c}{Z_c^2} = (-\frac{U_c}{Z_c} - q + ry) - x(-\frac{W_c}{Z_c} - py + qx) \tag{5.3}$$

$$v = -\frac{V_c}{Z_c} + V_c\frac{W_c}{Z_c^2} = (-\frac{V_c}{Z_c} - rx + p) - y(-\frac{W_c}{Z_c} + qx - py) \tag{5.4}$$

It must be stated that models for stereo vision and cameras that possess a wide angle view lens, acquire different modelling and estimation techniques. The former model can be characterized as a perspective model. If we had not assumed a viewed plane, an omnidirectional or spherical model would suffice, as explained in [40].

5.2.1. OF parameters

Now that the camera's ego motion has been linked to the perceived OF on the image plane, the actual ego-motion can be determined. Depending on the environment and viewed scene, a generally complex computation is required to determine all depths and rotational and translational speeds of the camera. It would require heavy iteration or optimization processing, which is not suitable for small-scale robotic applications. Instead, depending on the application, simplifications or other sensors can be used in order to determine the exact OF faster and more precise.

For general applications of OF, it is beneficial to separately measure the rotational rates (p, q, r). Including these rates in equation 5.3, will only let translational flows remain as the observed states in (u, v). This is called de-rotation, as ego-rotational movements are isolated from determining (u, v). The previous means the absolute position and OF of visual features on the retina or sensor would be known if an Inertial Measurement Unit (IMU) is available. For instance, in [41] [42] [43] [21] a rate gyro is added to the sensor suite that defines the unknown rotational speeds in order to determine ego-motion from OF.

Using OF for controlling a landing on flat surfaces allows us to implement another simplification. Since a planar and horizontal surface is assumed, all points are interrelated. The surface must possess some sort of static texture as OF can not be tracked from smooth surfaces. If the landing is purely vertically controlled and the camera is pointing downwards, the following derivation is possible.

As shown in [44], with the latter assumptions, the OF vectors (u, v) in the image plane are:

$$u = \frac{(-U_c + xW_c)}{Z_c} \tag{5.5}$$

$$v = \frac{(-V_c + yW_c)}{Z_c} \tag{5.6}$$

As the MAV will experience roll and pitch movements, the ground surface has an inclination with respect to the camera's frame (C). In this case the distance between the ground surface and camera (Z_c) can be expressed with $(Z_{c,0})$ (vertical distance to surface), $(Z_{c,X})$ (plane slope (X_c)) and $(Z_{c,y})$ (plane slope (Y_c)):

$$Z_c = Z_{c,0} + Z_{c,X}X_c + Z_{c,Y}Y_c, \tag{5.7}$$

which can be rewritten into:

$$\frac{Z_c - Z_{c,0}}{Z_c} = xZ_{c,X} + yZ_{c,Y}. \tag{5.8}$$

Since the height $Z_{c,0}$ is between the floor and $(x, y) = (0, 0)$, $(Z_{c,x})$ and $(Z_{c,y})$ are written as:

$$\alpha = -\arctan(Z_{c,x}), \;\; \beta = -\arctan(Z_{c,y}) \tag{5.9}$$

We can scale the velocities with respect to $Z_{c,0}$:

$$\omega_x = \frac{U_c}{Z_{c,0}}, \;\; \omega_y = \frac{V_c}{Z_{c,0}}, \;\; \omega_z = \frac{W_c}{Z_{c,0}}. \tag{5.10}$$

Consecutively, combining all parameters leads to the expression:

$$\begin{aligned} u &= (-\omega_x + x(\omega_z)(1 - x(Z_{c,x} - y(Z_{c,y}) \\ v &= (-\omega_y + y(\omega_z)(1 - x(Z_{c,x} - y(Z_{c,y}) \end{aligned} \tag{5.11}$$

When the inclination of the camera is unknown, the vertical velocity can not be determined. In other words: without knowing $(Z_{c,x})$ and $(Z_{c,y})$, ω_z can not be determined. As [45] puts it, a chicken-and-egg problem is created since the same OF measure might indicate different speed/distance ratio's. However, if the ventral flows ω_x, ω_y are kept small, the perceived optical flow only is accountable to ω_z [44].

$$\frac{\partial u}{\partial x} = \omega_x Z_{c,x} + \omega_z \approx \omega_z, \;\; \frac{\partial v}{\partial y} = \omega_x Z_{c,y} + \omega_z \approx \omega_z \tag{5.12}$$

As divergence (D) is expressed as

$$D = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}, \tag{5.13}$$

this means that $D = 2\omega_z$. Additionally, time-to-contact (TTC) τ can also be deduced from divergence, namely: $\tau = \frac{Z_c}{W_c} = \frac{1}{\omega_z}$. Knowing the time interval between images, when calculating time-to-contact, is required to transform it to actual seconds.

Although object detection is not the scope of this research, many similarities exist between OF-based navigation and object detection. Most object detection algorithms use the anomalies caused by objects in the OF field as an object detection method. When a MAV is moving forward in a purely translational motion, the FoE is regarded as a singular point from where OF expands. It indicates the course of the MAV and can therefore be beneficial for navigational purposes. The FOE can be determined using different methods. An iterative scheme in [42] uses negative half-planes to calculate and score a potential FOE area. [46] determines the FOE of every object in the frame by averaging the OF vectors intersection points. Knowing their future paths allows for object avoidance.

As explained previously, the actual height and horizontal distance to objects and planes is important to scale the other OF parameters. Knowing a MAV's altitude helps determine the forward speed and vice versa. Another solution could be stereo vision. By adding a second camera, a depth estimate can be made, which can provide translational velocities as done in [47].

5.3. Vision based flight control

Now that we know how to interpret OF parameters for different camera configurations and settings, a closer look is taken at how to use such features for control and navigation strategies. As previously mentioned, it is known that insects make use of parameters deduced from OF to perform certain flight manoeuvres. The following chapter will investigate how these parameters can deliver a suitable basis for control purposes.

Two important visual observables for (3D) landing are ventral flows and divergence. As explained before, height control is not possible with zero ventral flow, since they are coupled 5.12. Both ventral flow and divergence rate are known to be kept constant during landing of honeybees [31]. Honeybees also decrease their altitude when their forward speed is decreased by for example headwind.

In [48], a tethered rotor-craft performs landings on a moving platform. Both height and forward speed are controlled for with an OF regulator. Height is controlled with ventral flow while forward speed is regulated by means of pitch which affects the amount of horizontal thrust generated. Equal visuomotor properties are observed to that of insects and birds.

Often, a height approximation is given to MAV's to scale the perceived OF. Instead of using height, [49] integrates OF over a certain angle of a sphere to identify the scaled velocity. The OF distribution over a spherical lens projection can give a comparative indication of the distance between the vehicle and surface. This is used as a control variable for hovering and landing. The quadcopter is able to land on a moving target.

Assuming no ventral flows are present and thus only controlling for divergence, pure vertically controlled landing is possible. Several examples exist of divergence controlled landings. Differences are present in the type of controller or extraction method for divergence, but most of these applications adhere to an approximate bio-inspired landing strategy of maintaining a constant divergence during landing [44]. The authors in[39] evolve multiple controllers that are different types of NNs for a divergence-based landing. The new controllers are compared to a certain baseline controller. The newly evolved controllers show promising results, including strategies that at first seem less apparent.

5.4. Event based Vision

In order to improve speed and accuracy properties of OF-based flight control systems for MAV's, the measurement speed and accuracy of estimating OF must increase too. Frame-based camera's capture a viewed scene at a fixed temporal interval that is independent of the viewed scene dynamics. Pixels measure a continuous observed brightness of a certain scene or motion. For improving frame-based OF estimation, this would mean that the number of frames per second should increase and that the algorithms determining OF, must become more accurate. This generally increases the computational loads, which in turn decreases their usability for real-time robotic applications.

In contrast, event-based camera's output are dependent on the observed scene, since visual information is captured the moment brightness shifts are observed 5.2. The communicated information are asynchronous events that indicate positive (ON event) and negative (OFF event) changes in the log intensity at each pixel's location. Event-based camera's [3] possess a temporal resolution of $1s$ with a latency of only $1ms$. Since redundant scene information is not captured and processed, a high sample

rate is possible. For a frame-based camera to match the temporal resolution of 1 s, the fps should be multiple thousands of fps. The sparse and asynchronous event-based output reduces processing power and memory requirements, which are beneficial for high speed and low power MAV's.

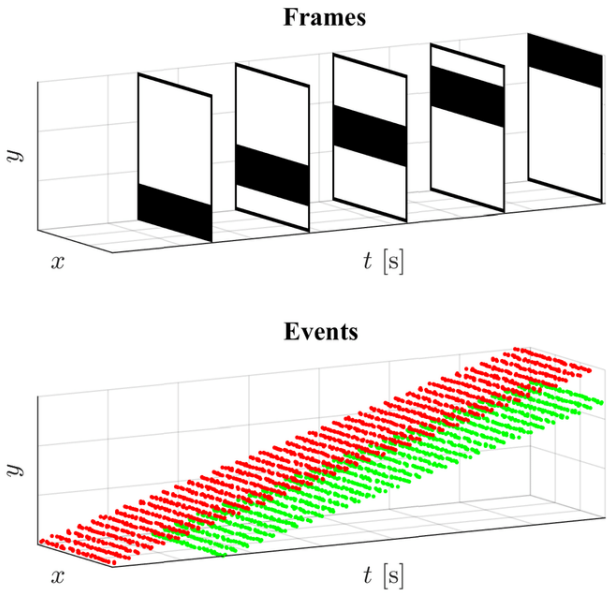


Figure 5.2: Illustration of the observed difference between frame and event based camera's. As the black bar in the image moves up, event-based camera's only respond to the leading and trailing edge of the black surface, since brightness changes are observed here. The green and red colours represent polarity of a positive and negative change, respectively.

The additional temporal dimension to visual measurements can unlock many potential possibilities for robotics, since more information is encoded using event-based cameras. This type of encoding of visual information is biologically inspired by the spiking nature of visual pathways. As shown in [50], any visual system of mammals and insects inhibit a visual system that perceives spatio-temporal variations of brightness shifts at retina level. The retina possesses light-sensitive neurons that absorb and convert light into electrical signals. These signals are sent to the ganglion cells, after which the activity is encoded into temporal sequences of discrete spikes. After, larger networks decode this raw information of observed brightness patterns to visual cues, in order for the brain to estimate ego-motion or perceive the environment. Event-based vision approaches its biological counterpart to visual systems, showing us at which great precision, speed and efficiency such type of information can be utilized.

In contrast to frame-based cameras, where every pixel records a brightness value at set time intervals, event cameras pixel activity is driven by light intensity changes. Every pixel, on a DVS, reacts to changes in log photocurrent $L \doteq \log(I)$. An event $e_k \doteq (x_k, y_k, t_k, p_k)$ is triggered at time t_k and pixel location (x_k, y_k) when

$$\Delta L(x_k, y_k, t_k) \doteq L(x_k, y_k, t_k) - L(x_k, y_k, t_k - \Delta t_k), \tag{5.14}$$

has reached a certain temporal contrast threshold of C :

$$|\Delta L(x_k, y_k, t_k)| > C. \tag{5.15}$$

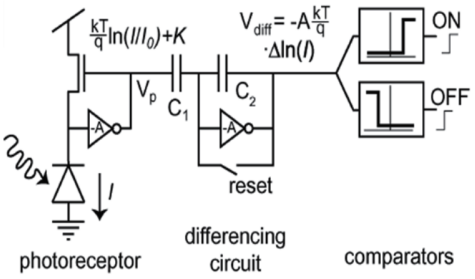
A spike is transmitted when this brightness shift exceeds the threshold in 5.15. These spikes are transmitted using an Address-Event Representation (AER) readout as in [3]. The messages include a pixel's position, a timestamp and polarity.

5.4.1. Event based camera's

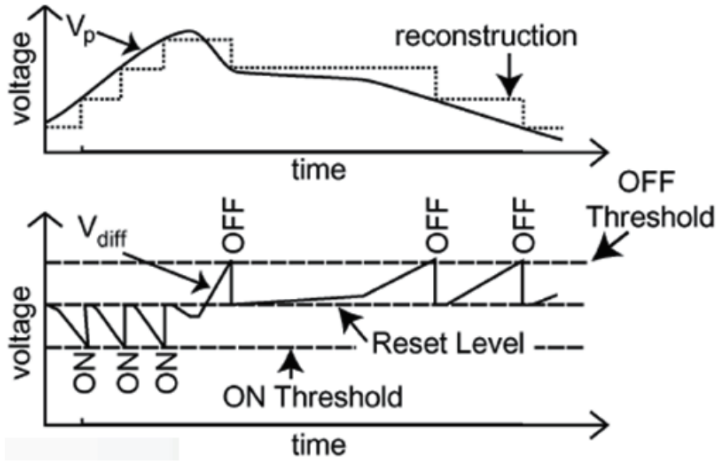
As mentioned in the section introduction, event camera's respond to an observed brightness change, as seen in figure 5.2. While the data flow for outputs of frame based cameras are constant, event-based camera data flow will increase if faster motion is observed.

Although event-based processing allows a more granular view of motion and can distinct fast motion accurately, the light intensity should not vary too much. There exists a cut-off frequency at which cycles in brightness shifts can not be captured. Additionally, while a frame-based camera aliases for frequencies above the Nyquist frequency, a DVS, due to the continuous time response, does not. Depending on the chip and hardware, the AER messages can become saturated. The latter does not directly affect any physical properties of the measurements, but has influence on the times at which the messages have been sent at.

(a) DVS camera



(b) DVS pixel



(c) Reponse according to fluctuating light intensity

Figure 5.3: Image adapted from [3] (a) A DVS with it's lens. (b) Schematic overview of a DVS pixel, which reacts to shifts with fixed-size changes of log intensity. (c) Shows the output in response to changing light intensity.

The first commercially made available event camera is the DVS [3] and is still commonly used [51]. The camera features a 128x128 pixel grid operating at an intra-scene dynamic range of 120 dB and possesses a s timing resolution and a latency period of 15 s. The continuous-time photoreceptors on the silicon retina are coupled to a readout circuit, which is reset each time the pixel was sampled. Basic properties of the camera and pixel working mechanism are found in figure 5.3

DVS is a conventional event-based camera that only outputs witnessed brightness changes. However,

some vision applications require more than that. Two types of event camera can be identified, namely cameras that only output brightness shifts and cameras that output the absolute brightness. Many applications need to know this form of static information in order to make a distinction between the same motion in a different setting. The Asynchronous Time Based Image Sensor (ATIS) [52] outputs both by implementing two pixels side by side, namely a DVS pixel and a traditional pixel reading out absolute intensity. In order to output static and dynamic readings simultaneously, a trigger resets the capacitor, which ensures no absolute light intensity is outputted if a brightness shift has not exceeded the intensity difference threshold. The mechanism is dependent on the absolute brightness of the scene for discharging the capacitor. The brighter the light, the faster the capacitor discharges and the faster the DVS pixel is able to respond to witness brightness changes. A downside of this mechanism is that, because of its discharge dependence on absolute brightness, its efficacy is compromised in darker scenes. The Dynamic and Active Pixel Vision Sensor (DAVIS) [53] combines an active pixel sensor with a DVS pixel. For DAVIS, the photodiode is shared between the two pixels. ATIS on the other hand, introduces a new pixel side by side which decreases the resolution of pixels per area. Another interesting event-based camera alternative for MAV's is the embedded DVS (eDVS) [54] that excludes an USB output connection and directly connects to a microcontroller for weight and scale benefits. Where the DVS weighs 120 $grams$, the eDVS only weighs 16 $grams$.

5.4.2. Event processing

A granular and high frequency event stream requiring low processing power is favourable for different real-time robotic applications. The following element to consider is how to extract meaningful features as efficient as possible for the task considered. Due to the granular and high frequency data flow out of a DVS, many applications require a transformation of the events to be able to extract features. Depending on the task ahead, certain representations and subsequent processing methods are suitable. A key parameter is the latency that is induced by grouping the temporal or spatial element of events. The following will first discuss event representations and after discuss further processing aspects for certain objectives.

The table below gives a brief overview of the mostly used event representation, including a short description as given in [55].

Table 5.1: Brief overview of all event representations as in [55]

Representation	***Notes***
Individual Events	The events do not undergo any form of pre-processing. Here, an event $e_k \doteq (x_k, y_k, t_k, p_k)$ is used by event-by-event filters and SNNs. For such methods, the earlier activity of the same pixel and close pixels is important for future output.
Event packet	Neurons are grouped (mostly neighbouring neurons) by either their temporal or spatial properties, and a single aggregated output is formed.
Event image	The events receive a simple conversion (averaging or counting events) before being projected on a 2D plane. This allows more conventional algorithms to process event-based data. However, sparsity in an area can not be viewed, and the converted image is highly sensitive to the absolute number of brightness shifts witnessed.
Time Surface	A time surface representation is a more elaborate version of an event image where the only pre-processing mechanism in place is to display intensity of a viewed scene. A higher value corresponds to a more recent brightness shift, which, in a basic sense, takes into account prior activity of that pixel.
Voxel Grid	A voxel grid represents a 3D histogram of events. Temporal information is more precise, since a voxel shows the duration of events, polarity and spatial coordinates.

Now, we should question how to process the above-mentioned even interpretations. Manipulation of the even data flow has different phases, such as input adaptation, feature extraction and output generation. The processing methods are constrained by the feature extraction algorithms and hardware platform required. Event processing techniques can be characterized by different algorithmic properties, such as event-by-event-based methods or methods for grouped events. Event-by-event-based methods mainly deal with the initial pre-processing tasks of noise reduction, low-level feature extraction and filtering. Filters

essentially perform a grouping task of smoothing asynchronous input and aggregating multiple event sources. The use of Kalman filters or SNNs, should theoretically lead to a more accurate determination of OF, since the grouping mechanism in filters is tuned to a certain temporal or spatial extent. Time surfaces, being grouped events, allow interesting further interpretation as scene edges become apparent. Voxel grids on the other hand, require more memory but attain a higher level of preserving temporal information. Deep learning applications initially favoured aggregated events as input, since the viewed motion or shape can easier be linked to a less noisy input sequence. Currently, more emphasis is put on algorithms that, in a biologically plausible fashion, extract features from individual events.

5.4.3. Event-Based Optical Flow

As mentioned previously, computer vision algorithms that use conventional frame-based cameras are constrained by the brightness assumption. The spatial and temporal derivatives of two successive frames are calculated or optimized for, for flow estimation. As is apparent for current OF estimation algorithms, edges are used as reliable OF estimation [33]. This makes EB OF estimation more attractive, since edges are the visual cue that EB camera's react to. In addition to the high speed and granular temporal properties, high speed flow can accurately be tracked. Furthermore, conventional computer vision have difficulties with large inter-frame displacements of tracked features, as the optimization framework must perform more iterations for equal accuracy. Motions overlapping will cause irregularities for conventional OF estimation techniques, while event-based sensors do react accordingly to such scenes. Event-based visual processing is a relatively new area of study and therefore a benchmark for an OF estimation algorithm, still has to be determined. Unfortunately, the separate events alone from a DVS do not allow for flow estimation and are in need of a certain aggregation function to understand the interplay between witnessed motion.

As EB camera's have no access to information such as absolute pixel intensity, visual features are hard to identify and subsequently track. Since much computationally expensive processing is necessary for identifying such features, most algorithms do not distinguish the events received from a DVS.

Early versions include simple adjustments of current algorithms, such as [36], to fit the event-based sensors. These gradient based methods use spatio-temporal derivatives and the brightness constraint [33] to estimate OF. In order to grant continuous flow, certain spatio-temporal properties must be conceded. The first implementations of event-based OF consist of extracting a continuous temporal derivative of brightness [56]. By replacing a single event by the sum of event polarities at a pixel location, an adaptation of the frame-based Lucas-Kanade tracker is possible. However, as the derivative is calculated using sparse events, the estimations turn out noisy and unreliable for further OF estimation. The approach in [57] also sums polarity values, but tries to minimize the inconsistencies by employing the second derivative instead of first and second derivatives. Due to the high frequency, the moving fixed time window must be accurately determined according to a scene. Else, the difference in witnessed intensities will lead to poor OF estimation. Additionally, the total number of events within a certain time period are too low to give precise estimates at a high frequency.

The following methods are based on viewing incoming events $e_k \doteq (x_k, y_k, t_k)$ of a moving edge as a surface. The surfaces created by incoming events can be fitted with planes that estimate the according flow orientation and amplitude. No additional aggregation function is necessary to calculate temporal or spatial gradient. Planes can be solved for by using Least-Squares optimization methods.First [58] proposed estimating normal flow from local spatio-temporal data. In order to improve accuracy, some assumptions and constraints must be imposed. For instance, as planes approach pure vertical or horizontal surfaces, the exploding or vanishing gradients must be dealt with. By introducing a threshold angle for the fitted planes, outliers of flow estimations can be omitted, as done in [57]. In [51], the number of plane parameters are reduced to perform more efficient optimization iterations. Additionally, by assuming a small constant velocity, the flow estimation is regulated against noisy events and produces stable estimations. [59] introduces a method that uses an IMU to correct OF induced by ego-motion. By filtering out all planes 'created' by the camera's movement, only the planes remain that are produced by objects and motions in the scene.

Filters can be applied at different stages for OF estimation. Simple filters as the Savitzky-Golay (SG) filter, can be implemented to improve the estimation methods mentioned above. The SG filter increases the signal-to-noise ratio by fitting a low-order polynomial with linear least squares. This filter can be applied to moving average functions (for capturing intensity) and local plane fitting. This filter does not

necessarily induce more computational complexity and time since a system of equations does not have to be solved as for Lucas-Kanade [36] or surface fitting algorithms [58]. The other class of filters are direction sensitive filters. These filters yield directional motional sensitivity and improve OF estimation for certain scene properties. Filter banks in [57] are tuned to certain directions and flow velocities, for which OF is determined. Determining the exact magnitude of OF using direction sensitive filters can be problematic, as the performance is mainly dependent on the configuration of filters. As determining OF becomes more refined, the range of scenes for good OF approximation becomes narrower. As highly textured areas can introduce estimation irregularities, high spatio-temporal frequencies, that are induced by texture, are minimized leading to more accurate OF estimation [60]. However, the additional computational loads do not allow smooth real-time operation.

Due to the continuous input flow necessary for Artificial Neural Networks (ANNs), frame-based cameras are ideal for feature extractions, since a synchronous image feed is provided. Taking into account established learning rules, ANNs have proven to be powerful for many vision related tasks. In order to use ANNs for an event-based stream, modifications must be implemented to allow ANNs to interpret the data flow. As ANNs necessitate a continuous input, the sparse flow must be modulated. Additionally, since conventional learning algorithms, such as backpropagation, require discretized parts being trained in episodes, the event stream must be divided into multiple temporal slices. This reduces the potential of extraction due to not fully using the asynchronous properties of even-based vision.

In [61], an encoder-decoder Convolutional Neural Network (CNN) architecture (EV-FlowNet) is used to match an event-stream with the equally observed scene in a greyscale representation. For OF estimation, a self-supervised learning method is implemented to learn to predict normal flow from events out of a DAVIS camera. A greyscale picture is used to build a ground truth loss function, meaning labels of such are unnecessary. The loss function consists of a photometric and smoothness loss. The photometric loss aims at minimizing the difference in intensity between the modulated event representation input, and greyscale image output. Later, in [62], EV-FlowNet functionalities are extended by means of an unsupervised learning scheme. The aggregation function interpolates discretized event volumes. This representation encodes a distribution of all the events in a certain spatio-temporal domain, which can be linked to the perceived motion blur present when decoding the distribution of events. Therefore, an unsupervised-learning signal is possible. The authors in [63] have created a framework that can convert models trained on a synchronous image feed to asynchronous models. Thus, by manipulating input data, the asynchronous and sparse properties of event-data can be maintained. A local update function is introduced that shapes the event signal by updating an activity measure at every individual pixel. The local update rule is dependent on the location of the pixel and the pixel's past activity.

Evidently, the characteristics of SNNs match the spatio-temporal properties of an event-based sensor. Theoretically, SNNs should be able to reach more potential since even-data does not have to be encoded in a continuous and synchronous fashion. Many SNN applications for OF are biologically inspired, as mammals and insect's visual systems possess processing characteristics similar to LIF or AEIF neurons.The authors in [64] present a convolutional spiking architecture with properties of a filter that shows direction and speed selectivity. Feature extraction is possible through an unsupervised learning rule that links spiking activity and connectivity patterns to certain kernels that represent a spatial feature. These features allow local motion perception and global ego-motion to be estimated. Such features are useful for MAV navigation.

In [65], spatio-temporal filter are implemented with certain delays to capture motion. A total of 8 speed and 8 direction sensitive filters are used. Each neuron has unique purpose, therefore no overlap of functioning is present. The introduced time-delayed synapses do increase the need for longer on-chip storage of data. In the works of [66], a discretized input representation (fine-grained in time) is presented that preserves the spatial and temporal information of events for SNNs. The encoder-decoder architecture uses SNNs and ANNs for its deep hybrid architecture. The vanishing spike problem regards the disappearing activity in a network that decreases learning opportunities. By realizing ANNs for the decoder part, OF prediction learning can still make use of the beneficial learning traits ANN's have.

6

Visuomotor characteristics of insects for flight control

The following chapter will discuss the visuomotor configuration and corresponding reflexes of insects. In order to understand the flight control properties that insects have, the contribution of the organisation of motion detection and processing is analysed. This includes labelling and understanding the visual systems of flies and how the perception and processing is coordinated. As we know, insects base their guidance and navigation on OF. While OF only gives us information regarding a speed and distance ratio (chicken-and-egg problem [67]), insects are still able to navigate through unpredictable environments. Not only is the visual processing responsible for this, but also the receptive field organization. The receptive field organization initially dictates the sensitivity and selectivity by means of interpreting motion differently across the eye's surface. Explanations will consider the Drosophila, whereas if another type of fly is not referred to explicitly.

6.1. Neuronal Diversity in Visual system

The processing of motion starts in the retina. After these brightness patterns have passed through an array of photoreceptors, the information is processed by three neuropiles. These neuropiles are retinotopically arranged and consist of the Lamina, Medulla and Lobula. The flies visual processing system is hierarchically ordered, which means that more complex processing is applied sequentially at deeper layers of the brain.

Retina The first perception of a scene is captured by light sensitive cells. The resolution of the retina is according to the number of ommatidia present on the surface that are spread over a certain angle width. Sometimes these ommatidial rows are not evenly distributed over the retina. The female blowfly Calliphora has a two times higher resolution in the frontal field than it has in the lateral part [68]. Each ommitidium has overlapping photoreceptors that converge in groups of 6 into a single signal to the lamina (R1-R6 6.1B). Photoreceptors R1-R6 have found to be responsible for visuomotor behaviour [69]. It is thought of photoreceptors R7-R8 to be involved with colour processing [70].

Lamina The Lamina is the first optic ganglia that receives optic information from clustered R1-R6 photoreceptors. It essentially amplifies the signals from the clustered photoreceptors, but narrows down the dynamic range. It's functioning could be compared to a high pass filter [71]. Lateral connectivity and feedback connections appear to be present, though not as elaborate as the Lobula, since local signal processing prevails.

Medulla The Medulla acts as a low pass filter. It performs a processing step between the 'brushed off' high frequency signals from the lamina and Lobula, for the preparation of motion extraction 6.1A. Part of the photoreceptors (R7 and R8) are directly connected with the outer part of the medulla. [72] has recorded that orientation selective responses occur in different layers of the medulla. This can be regarded as the first processing step in extracting features.

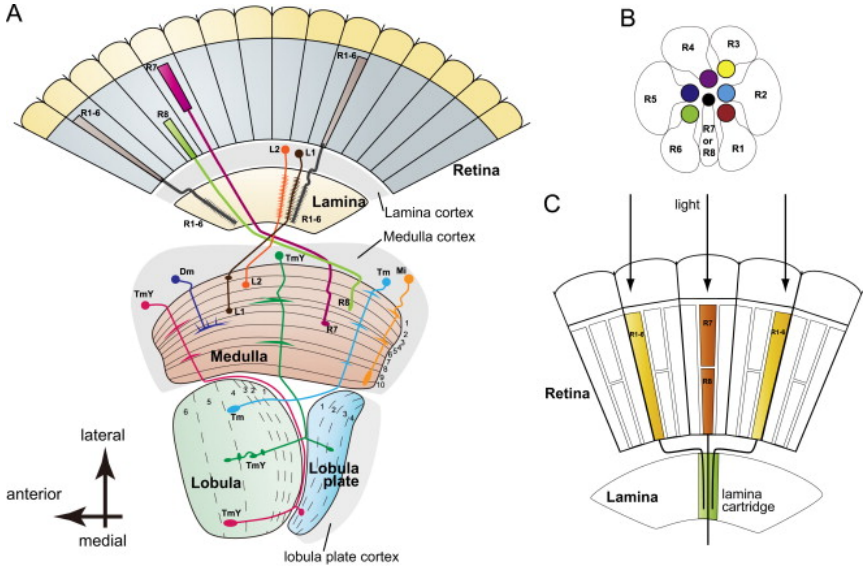


Figure 6.1: Visual processing system of flies

Lobula Complex As mentioned previously, the Lobula is the neuropile with the most lateral and feedback connections, which is necessary for recognizing and processing observed motion. The direction selective processing properties allow for integrative sensitivity. The Lobula complex contains two separate regions for flies, namely the Lobula and Lobula plate.

Lobula tangential plate cells As mentioned before, the Lobula can be divided into two: the posterior Lobula and the anterior Lobula plate. In the Lobula plate, Lobula Plate Tangential Cells (LPTC) exist that consist of wide field motion sensitive neurons. These neurons process a significant amount of visual information and form a crucial link, which is a relatively short neuronal connection, between the perceived motion and control of body movement. The LPTC is responsible for the link between witnessed motion and body behaviour. An important part of the visual processing abilities is explained by the different motion attributes that different neurons in the LPTC are able to detect and react to accordingly. The following characteristics (or combinations) exist for the approximately 60 tangential cells in the LPTC [73]: 1) Orientation selectivity 2) A response or shift in firing rate as a result of a certain motion 3) The location of where the output of the cell is connected to, being either to the hetero lateral, ipsilateral side or both and 4) Whether the responses increase with the increase of a visual pattern or are only tuned for small motions.

6.1.1. LPTC orientation

A population of 60 visual interneurons are present on the Lobula plate. Most of these LPTC's have extended dendritic branches that possess input fields of distinct areas of the retinotopic arrangement. In these dendrites a lot of preprocessing of the signals occur that are characterised as Elementary Movement Detectors (EMD). Again, 4 categories of pre-processing groups have been identified [74]: 1) Heterolateral LPTC's that receive inputs from a large range of the receptive field 2) Direction selective small field elements 3) Centrifugal cells that combine the motion inputs from different sources in the receptive field which is also linked with 4) Figure detection cells that receive small field motion elements from wide field area's.

The output of LPTC cells have been found to be orientation sensitive (hausen 1982) and consist of two groups, namely the horizontally and vertically sensitive cells. These cells have been linked with the perception of self motion, though some might argue that this direction selectivity already occurs in layers prior to the LPTC. The general alignment of the dendrites explains how this distinction is made.

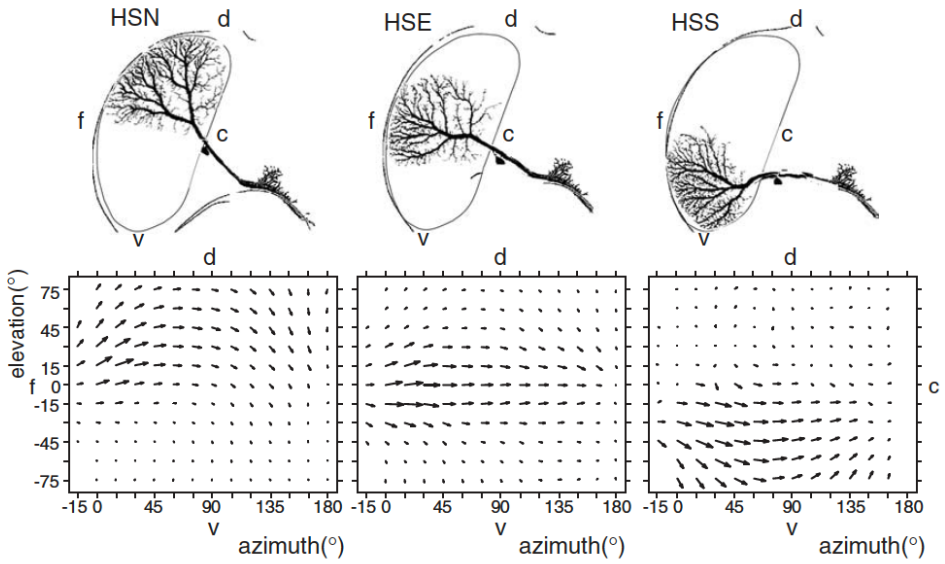


Figure 6.2: Picture adapted from [74] showing motion sensitivity across HS cells

Horizontal System Cells The Horizontal System (HS) cells can be divided into three groups namely the HS North (HSN), HS Equatorial (HSE) and HS South (HSS) group [74] 6.2. The HSN, HSE and HSS cover the dorsal third, middle third and ventral third respectively. The characteristics of these dendrites are not equal, since the HSS-cells do not react to back-to-front motion as HSN- and HSE-cells record this. The HSN-cell has found to be most active for motion slightly above the eye equator at an azimuth angle of $0 - 15\circ$ [75]. The HSN and HSE group receive additional rotation signals from other (VS) heterolateral cells that are thought to be used for identifying rotational movements.

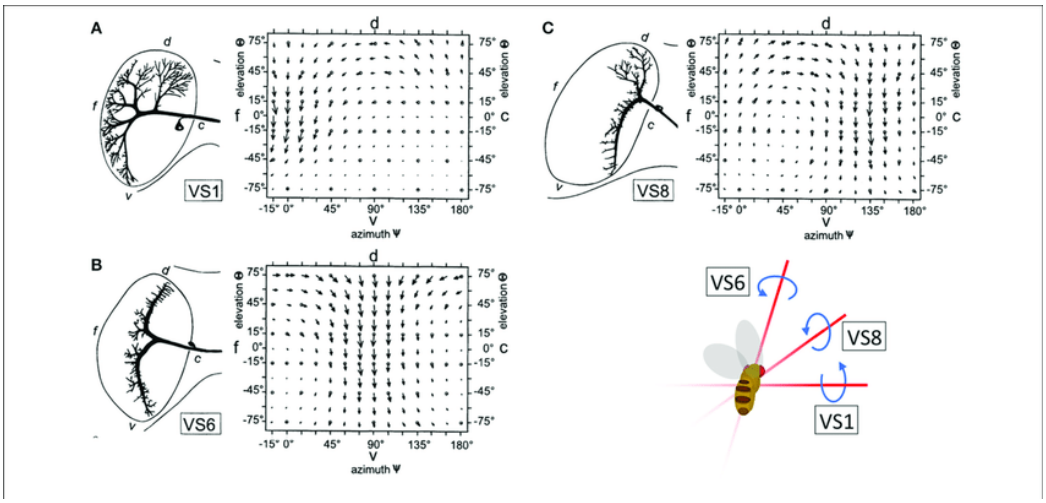


Figure 6.3: Picture adapted from [76] showing motion sensitivity across VS cells and corresponding selectivity to observing rotation for different axes

Vertical System Cells The Vertical System (VS) cells has 10 branches, starting with VS1 at the most anterior position up until VS10, which is located at the most posterior area of the receptive field. As shown in figure 6.3, VS1 is especially sensitive to vertical downward motion (pitch) in the frontal field. Additionally, VS1 has some sensitivity for back to front motion due to the extended dendritic branches present in the most posterior field. Other VS-cells cover right-handed and left-handed rotations between the axes located in pitch and roll [77].

Centrigual Horizontal cells The Lobula plate contains two Centrifugal Horizontal (CH) cells per hemisphere, namely the dorsal CH (dCH) and ventral CH (vCH). Some do not regard CH-cells as a separate system, since the CH-cells are dependent on HS-cells and not the other way around. The CH-cells are more reactive to rotational than translational stimuli and are known to combine information from both eyes [73].

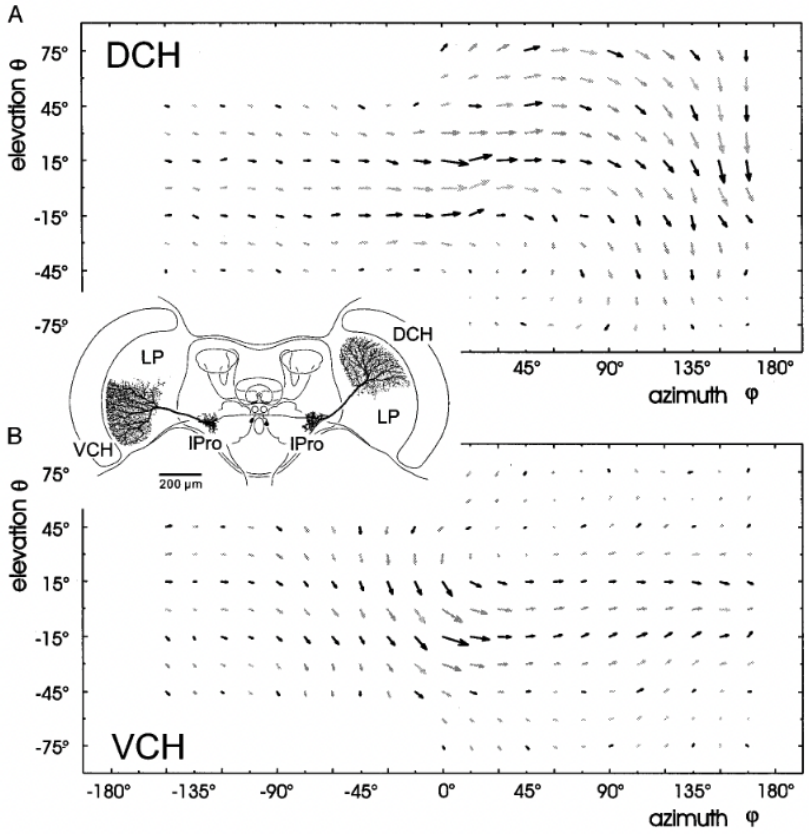


Figure 6.4: Picture adapted from [78] showing motion sensitivity across CH cells. Note that in the caudal equatorial part of the dCH response field shifts from horizontal to vertical. Although the image only depicts one side of the directional sensitivity of the eye, symmetric response for both eyes can be assumed

6.2. Processing Characteristics and Sensitivity

After assessing the directional motion sensitivity, we could question how the directional sensitivity effects the flow determination for ego-motion and subsequently navigation purposes. Not only does the distribution of local sensitivity match certain OF fields, but within local motion sensitives, a distinction is made in rotation- or translational-induced OF. As [74] suggests, some local motion LPTC's have been optimised to distinguish two types of OF, which are rotation- and translation-induced OF. Decreasing motion sensitivity for translational flow in the ventral receptive field makes sense, since this area would mostly be dominated by translational-induced flow. The distance between the insect and the ground is much smaller than the horizontal distance to distant objects. If roll induced OF would be sensed by the ventral receptive field, this OF would be dominated by translational-induced OF. The same accounts for sensing roll-induced OF. Detecting ventral flow on the vertical side plane from the insect would be dominated by roll induced flows. Translational flow estimation also shows higher sensitivity in ventral directed areas, as seen in the right most image of the HSS-cell 6.2. This means that the upper 2 two thirds are responsible for observing rotation induced flow. A mechanism is in place that compares roll induced OF, on the side where observed vertical flow is dominated by roll movement. If there is a match, roll induced ventral flows are filtered out to improve other navigation tasks. It is important to note that the HSS cells participate less in motion integration, linking OF to a wider field of range. This allows us to deduce that the receptive field below the equatorial horizontal line, predominantly detects ventral flows. HSE and HSN are responsible for sensing

yaw induced self-motion. Additionally, it is believed that CH cells seem to indicate banked turns [78]. The most important take away, is that yaw motion is not extracted from lower ventral regions. In [79] it is found that can better regulate control for expansion and contraction patterns than rotational patterns. This is due to the higher variance measured for wing torques. Flies are more sensitive to translational patterns than rotation.

After analysing the directional sensitivity, it is interesting to observe that many vision applications do not highlight that the OF estimation is effected by the camera orientation. In order to improve the quality of sensed OF, the eye is selective to seeing certain OF directions at different locations on the retina. Figure 6.5 confirms this by showing the preferred rotational axis

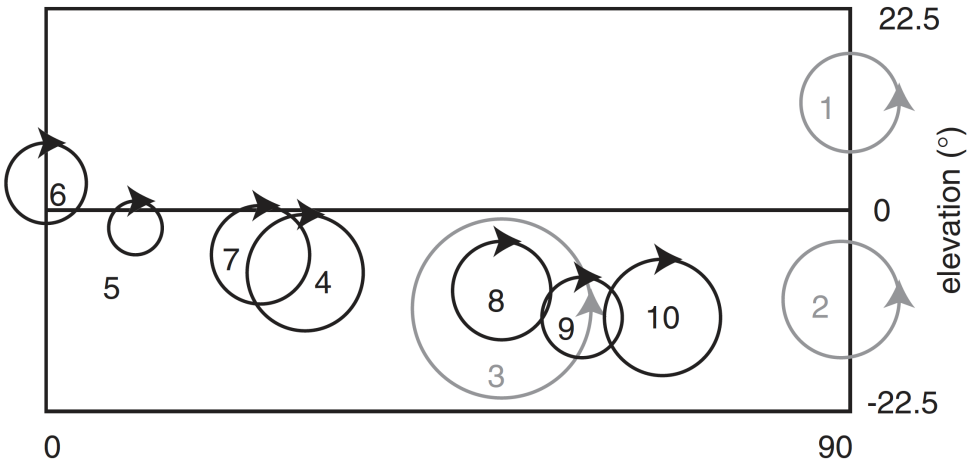


Figure 6.5: Picture adapted from [74] using data from [77]. The figure shows the preferred rotational axes of the VS-cells in the visual system of the Calliphora. A cylindrical projection of the right visual hemisphere is assumed. The VS-cells in black are excited by right-handed rotation, and grey circels represent left-handed rotation. The same rotations can be viewed in the lower right corner of figure 6.3. The diameter of the circels illustrate the standard deviation of the preferred axis of rotation. The missing x axis label is azimuth $(\circ)$

7

Evolutionary Algorithms

Evolutionary Algorithms (EA) are a robust optimisation method for difficult search environments. This is mainly due to their flexible representation of decision variables and performance evaluation. EAs are useful when the system dynamics are highly non-linear and the stochastic environment is poorly understood. Most optimal control problems rely on a direct or indirect gradient to converge to a local optimum. When the ill-behaved cost landscape or system dynamics are mathematically hard to characterize, EAs are a robust tool for delivering a converging set of decision parameters. The latter characteristics have led to many applications where an EA has been used for developing controllers [80]. EAs also allow for relatively easy off-chip simulations, where hardware or other operational constraints can be implemented.

As there are many EAs with often additional diverse variations, the treated algorithms in this chapter will only include algorithms that have fundamental differences in the mutation, selection and cross-over operators. Variations mostly impose slight modifications to an operator, but these characteristics will not be treated. First, the basic EA will be explained. Multi Objective Evolutionary Algorithms will follow, after which Evolutionary Strategies will be discussed.

7.1. The Algorithm

Natural computing is an encompassing term which describes modelling methodologies inspired by naturally occurring phenomena [81]. EAs belong to a class of probabilistic adaptive algorithms based on the simplified dynamic principles of biological evolution [82]. An EAs approach to solving or optimizing an objective function, happens by changing the decision variables to a certain extent and attempting a modified set of decision variables to the problem domain. A certain set of different decision variables is referred to as a generation. A generation evolves over time, thereby changing the decision variables, while the performance of every set of decision variables is kept track of. Further evolutions are based on the performances tested by generations in the past. As for the biological factor when considering EA, a ”Survival of the fittest” mentality is enforced. EAs belong to meta-heuristic search methods that discovers a new set of parameters based on experience gained from previous sets.

The three main attributes of an EA, whose operators can be adjusted, are the capacities of selection, mutation and recombination [82]. Populations of different decision variables are compared and depending on their performance removed, retained or adjusted via mutations and cross-overs. A mutation in the context of an EA means that a decision variable configuration is based on a previous version, while a cross-over transformation combines either mutated or other versions of a set. Other global parameters which can be altered for a different kind of preferred performance are the randomness in sampling methods, establishment of population or the actual objective function itself. In order for an EA to perform accordingly, the following properties are necessary for such frameworks: A design or representation of the parameters evolved, the performance of a decision maker that must be tracked and an operator which can create an offspring.

The reason that EAs are robust is that the exploration and exploitation features of the search algorithm are adequately balanced. If this ratio is biased towards exploitation, for example, only local optimum solutions might be found. [83] highlights that exploration and exploitation traits can not be assigned to a single EA operator. A common belief is that exploitation is achieved by mutation and exploitation is done

by selection. However, [83] points out that this is a misconception and that it is hard to address these features exactly.

7.2. Multi Objective Evolutionary Algorithms

Up until now, the explanations given assumed a single objective to be optimized, however gross of control problem domains focus on optimizing or solving for multiple objectives: Multi-Objective Evolutionary Algorithms (MOEA). These objectives are often conflicting, which means that improving one objective might deteriorate the other. A single optimal solution does not exist for all objectives simultaneously. The relative performance against each objective must be compared. The Pareto front is composed of solutions that compares the scores of different sets between different objectives. By plotting their relative performance, a sufficient trade-off solution can be chosen. The set of parameters leading to the least conflicting objectives is favorable. Often a substitute function is formulated that turns the vector optimization into scalar optimization. Though, these classical methods that create a substitute problem have a harder time identifying the conflicting objective traits.

7.2.1. NSGA-II

One of the most well known MOEA algorithms is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [84] and it's prior version, NSGA [85]. The core idea of NSGA is identifying the different Pareto fronts present among the individuals. The non-dominated individuals are classified according to their performances and withdrawn of the next ranking cycle. This process is repeated until a sufficient number of fronts are identified. A random dummy value is given to the individuals present in different Pareto fronts. The individuals located in a better performing front will always receive a higher score than individuals ranked lower. A stochastic operator assigns a certain reproduction probability according to this dummy value.

In comparison to NSGA, NSGA-II has decreased computational complexity, which increases the number of possible individuals and possesses a more elitism favourable approach. Non-dominated sorting might be unfavourable, since the elitist solutions to some objectives are not used to construct a new generation. NSGA-II addresses these issues by introducing a crowding measure and a binary tournament selection operator for ranking the individuals and increasing likeliness of good performing individuals for reproduction. Additionally, the computational effort decreases since much time is lost identifying multiple separate Pareto fronts. By only performing a single ranking computation, time is saved.

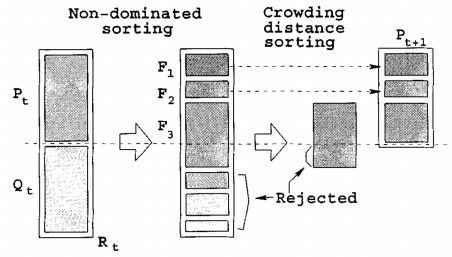


Figure 7.1: The figure is adapted from [84] showing the NSGA-II selection procedure.

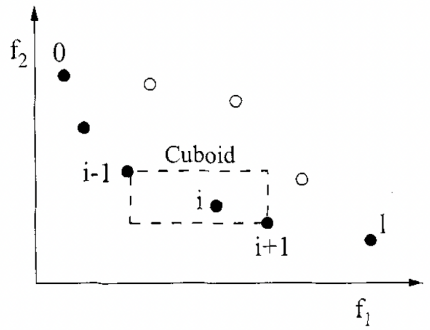


Figure 7.2: The figure is adapted from [84] showing the crowding measure analysis.

In [86], the NSGA-II algorithm was used to find appropriate control points for a trajectory defined by B-splines. By separating the path planner in a higher level offline and more precise online planner, trajectories were found for an UAV. The result proved NSGA-II to be suitable for finding suitable trajectories in a short time. The last part of the computational time would be used to refine the trajectory.

7.2.2. SPEA-II

The Strength Pareto Evolutionary Algorithm (SPEA) [87] uses a regular population and an archive of individuals. The archive allows good performing individuals to persist over coming cycles if a good scoring individual has a higher score than other individuals created in cycles after. It is useful to maintain an

individual that might not have the highest score overall, but performs well in a certain objective for future mutations. A fitness and strength value are awarded to every individual in the archive, which influences future reproduction probability. In the mating selection, phase individuals of the current mutation and archive are selected by a binary tournament. Consequently, this recombination and mutation replaces the current population. An important aspect of the archive is that non-dominated individuals are appended to the archive. If an individual is dominated or a duplicate exists, the individual is disregarded. SPEA-II [87] uses a fine-grained fitness assignment to the individuals in the archive, although now the archive size is fixed. When the archive can not be filled with non-dominated solutions, dominated individuals are added. The exact scoring difference between SPEA and SPEA-II can be viewed in 7.3.

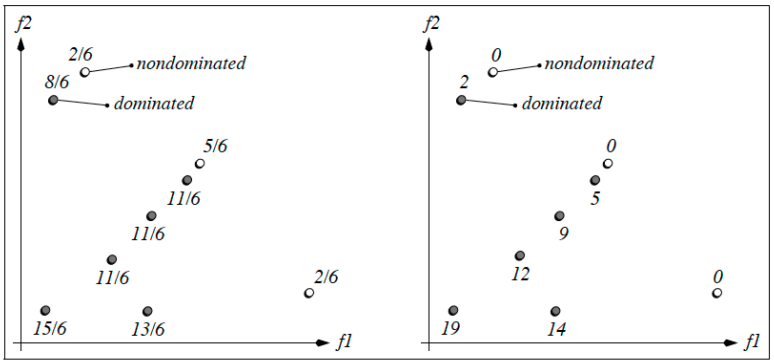


Figure 7.3: The figure is adapted from [87] and shows a difference in fitness assignment scheme between SPEA and SPEA-II. An exponentially less fitness value is given to individuals on lesser performing Pareto fronts.

7.2.3. Decomposition

The Multi-Objective Evolutionary Algorithm based on decomposition (MOEA/D) is a MOEA that decomposes a MOP into different scalar optimization sub-problems and optimizes these simultaneously [88]. Each sub-problem only uses objective evaluations of neighbouring sub-problems. This is a difference with NSGA-II since the non-domination sorting process requires the algorithms to know the evaluation of every generation. Decomposing the multi-objective problem to different scalar optimization problems can be done in different ways. The objective of each problem is a weighted aggregation of the objectives. A sub-problem i is a close neighbour to a sub-problem j if the weight vector looks similar. In the most simplistic set up of the MOEA/D, every sub-problem holds a single solution in its memory. This solution is the best solution encountered so far. As mentioned before, the idea is that the algorithms generates a new solution by only using solutions of it's neighbouring sub-problems. If the new individual scores better than the one before, the solution is adopted.

7.2.4. Differential Evolution

Differential Evolution (DE) is a population based direct search method. It is similar to a conventional EA, but the mutation operator perturbs current individuals with a scaled difference of randomly selected other individuals. The main advantage of such a mechanism is the excluding probability distribution that must be sampled from for producing new individuals, as done in most EAs. Another advantage of DE is that only 2 parameters need to be set, namely cross over probability and differential weight. The differential weight is multiplied by the difference between two randomly chosen individuals.

[90] used DE to find acceptable parameters for a SNN. Since an objective evaluation for this application takes 30s-4min, an efficient learning algorithm was necessary. An EA was chosen to search the discontinuous and multimodal optimisation landscape. DE has proven to be useful since the algorithm has led tot the best performance and had the least amount of evaluations necessary. Though it should be mentioned, as this is often the case, that a random search performs not too far off from EAs.

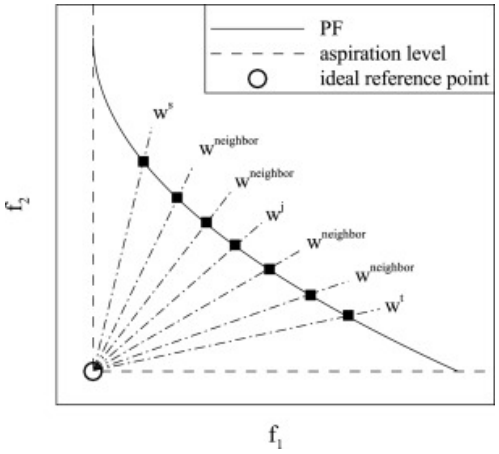


Figure 7.4: The figure is adapted from [89] showing the ideal pathway of solution on the Pareto front to follow.

7.3. Evolution Strategies

Evolutionary Strategies (ES) possess learning and adaptation features that have effect on the mutation operator of an EA. The difference between EA and ES is that these self-adaptation mechanisms depend on the course of step-wise improvements of the objective function.

The origins of ES do not lay in the effort of finding an optimum but came forth out of a concept that, using a set of pre-determined rules, would adjust variables automatically after every consecutive experiment [91]. Random perturbations of the variables and discarding of worse performing populations would lead to improving solutions. These random perturbations would later become Gaussian distributed mutations. All later versions of ES algorithms attain the latter two key concepts, which is slightly different from EAs. The performance of ES algorithms depend on the internal parameters, i.e. pre-determined rules, that influence the mutation strength.

A crucial difference to most EAs is that only newly generated populations are taken into consideration for the selection pool. This means that parents, also when outperforming all off-springs, are not present in the selection pool and are forgotten every iteration. For a condition of convergence to hold, the number of selected parents must be smaller than the size of the selection pool itself. This implicitly means that the optimisation will only occur successfully when at least one less performing solution is discarded for future mutations

The mutation operator is influenced by the trajectory in the fitness landscape. Here, a mechanism must be present that controls the variation operators, which are influenced by the performance trajectory. As explained before, a key feature of ES is self-adaptation. The parameters that are dependent on the objective function value and hence control certain statistical properties of the operators, are called endogenous parameters. These parameters can learn and evolve over iterations. Exogenous parameters are determined up front by the user and are kept constant during the evolutions. Furthermore, it is important that the mutation operator is unbiased. That being said, the scalable nature of ES algorithms are due to the continuous improvement scheme.

7.3.1. CMA-ES

For a wide spread of applications, the Covariance Matrix Adaptation ES (CMA-ES) algorithm has proven to attain advantageous convergence properties over other ES algorithms. A key feature is the scaling of the search space, increasing its robustness and reliability. Different versions have been developed over time, however only the most common version will be elaborated on. As pointed out in [92] the first version of CMA-ES had no real benefits for larger parallel applications since the performance for the number of function evaluations decreases linearly with increasing population. This version increases the adaption speed by introducing covariance matrix adaptation. The matrix calculation allows larger selection pools for less necessary evolutions.

The basic form (μ, λ)-CMA-ES selects μ sets out of the λ sized pool to calculate an offspring for the

$g+1$ generation by:

$$x_k^{g+1} = \langle \mathbf{x} \rangle_\mu^{(g)} + \sigma_g \mathbf{B}^{(g)} \mathbf{D}^{(g)} \mathbf{z}_k^{(g+1)}, k = 1, \cdots, \lambda. \tag{7.1}$$

where

$$\mathbf{B}^{(g)} \mathbf{D}^{(g)} \mathbf{z}_k^{(g)} \sim \mathcal{N}(\mathbf{0}, \mathbf{C^g}) \langle x \rangle_\mu^{(g)} = \frac{1}{\mu} \Sigma \mathbf{x}_i^{(g)} i \Subset \mathbf{I}_{\mathit{sel}}^{(g)} \tag{7.2}$$

$\langle x \rangle_\mu^{(g)}$ represents the centre of mass of the μ selected individuals of generation g and $\sigma_{(g)}$ is the step size. The random vectors z_k are randomly distributed with expectation zero and the identity covariance matrix, in order to generate an offspring for generation $g+1$ which is similar to calculating the centre of mass as before:

$$\langle z \rangle_\mu^{(g+1)} = \frac{1}{\mu} \Sigma \mathbf{z}_i^{(g)} i \Subset \mathbf{I}_{\mathit{sel}}^{(g+1)} \tag{7.3}$$

$\mathbf{C^{(g)}}$) is the symmetrical positive covariance matrix of random vectors $\mathbf{B}^{(g)} \mathbf{D}^{(g)} \mathbf{z}_k^{(g+1)}$. $\mathbf{B}^{(g)}$ has columns that represent normalized eigenvectors of the covariance matrix, while $\mathbf{D}^{(g)}$ is a diagonal matrix of which the elements are eigenvalues of $\mathbf{C}^{(g)}$. The surfaces of the probability density function $\mathbf{B}^{(g)} \mathbf{D}^{(g)} \mathbf{z}_k^{(g+1)} \sim \mathcal{N}(\mathbf{0}, \mathbf{C^g})$ created by the random vectors define the ellipsoid area where a new vector is drawn from. The exact equations for the adaption mechanisms are too detailed to be treated here, but for a more detailed description, please refer to [92]. It is important to know that the mechanisms consist of adapting the covariance matrix $\mathbf{C}^{(g)}$ and calculating an appropriate step size. The correlation between consecutive steps is used to update the $\mathbf{C}^{(g)}$ matrix and the step size uses the evolution path similarly but scales it with $\mathbf{D}^{(g)}$ to discard the dependence on the absolute size of the ellipsoid.

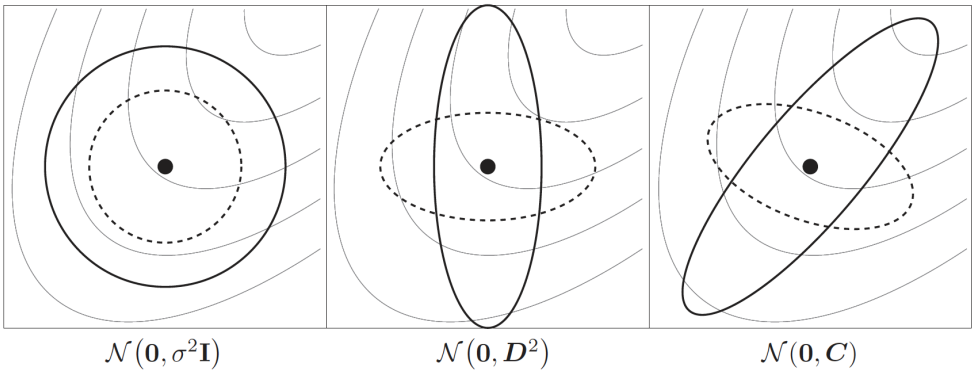


Figure 7.5: Figure shows the different sampling distributions in the CMA-ES algorithm. C is the positive definite full covariance matrix. Adapted from [93]

7.4. Natural Evolution Strategies

ESs are a type of EA that possess adaptive learning properties. These mechanisms, discussed before, take into account the trajectory of objective evaluations through the fitness landscape and adjust the mutation operator accordingly. Natural Evolution Strategies (NES) take it a step further by basing parameter adjustments solely on the stochastic returns of objective evaluations. Both ES and NES are regarded as black-box algorithms, since the generic optimization scheme allows finding correlations and control policies no matter the action frequency or delays (in the environment). Black-box algorithms are also regarded as direct policy search methods. Where Reinforcement Learning (RL) algorithms learn by perturbing the action space, allowing the agent to explore the environment, black-box algorithms don't need any exploration phase to understand the environment and produce adequate control policies. NES uses a natural gradient to update a parameterized search distribution in the direction of higher expected fitness [94].

The main difference between ES and NES, is the representation of the population and how the mutation and cross-over mechanisms perform. ES does not create a distribution for every single parameter. The following will discuss how a stochastic objective evaluation signal regulates the weight modifications. In 7.4, F is the objective function F that is acting on decision variables θ. The parameters in NES algorithms are represented by a population $p_\psi(\theta)$, where the distributions of parameters θ are indicated by ψ. A population with a distribution of sampled decision variables $p_\psi(\theta)$ is applied to maximize the averaged objective value $\mathbb{E}_{\theta\sim p_\psi}F(\theta)$. By perturbing the parameters, a distribution ψ is formed. The latter is regarded as stochastic gradient ascent, which is comparable to how RL algorithms utilize the stochastic objective evaluation provided by the environment.

$$\nabla_\psi \mathbb{E}_{\theta\sim p_\psi}F(\theta) = \mathbb{E}_{\theta\sim p_\psi}\{F(\theta)\nabla_\psi \log p_\psi(\theta)\} \tag{7.4}$$

The issue here is the possible non-smooth environment feedback or potential following discrete actions, resulting in ambiguous objective evaluations wherefrom no adequate gradient can be determined. In order to overcome this, the population distribution p_ψ is regarded as a multivariate Gaussian with mean ψ and covariance $\sigma^2 I$. Now $\mathbb{E}_{\theta\sim p_\psi}F(\theta)$ becomes $\mathbb{E}_{\theta\sim p_\psi}F(\theta) = \mathbb{E}_{\theta\sim N(0,I)}F(\theta+\sigma\epsilon)$. Therefore, the returned stochastic objective evaluation will conceivably be an adequate approximation of a Gaussian. Now the gradient function estimator for a stochastic gradient ascent becomes:

$$\nabla_\theta \mathbb{E}_{\theta\sim N(0,I)}F(\theta+\sigma\epsilon) = \frac{1}{\sigma}\mathbb{E}_{\theta\sim N(0,I)}\{F(\theta+\sigma\epsilon)\epsilon\} \tag{7.5}$$

Equation 7.5 can be approximated with samples. In order to calculate a gradient in the direction of the expected fitness, the population distribution p_ψ must be sampled. Where conventional EA and ES have some ranking mechanisms in place, followed up by a mutation operator, NES combines the fitness calculations to compute a gradient. The gradient is consequently multiplied with a learning rate to calculate the eventual parameter adjustment. As mentioned before, a major advantage of such learning rules is the independence of discontinuous environment dynamics or sparse actions of an agent. Additionally, the learning process/characteristics and performance are independent of simulation time in contrast to RL, where the learning is heavily influenced by the duration an agent spends in the environment. In other words, black-box optimizers are invariant to the frequency at which an agent acts in the environment.

In order to decrease variance in the distribution ψ, mirrored sampling can be applied. This mean that only pairs of perturbations are evaluated $(\epsilon, -\epsilon)$. [95] also mentions that it is useful to apply some form of ranking the stochastic returns. The latter removes outlier evaluations in each population to decrease the possibility of reaching a local optima in the early stages of learning.

Similar to RL and other optimization control problems, a stochastic return reward is provided. A difference is the performance evaluations out of these episodes/iterations for NES are used to estimate a gradient for each decision variable. For conventional backpropagation algorithms, the gradient is computed according to an error. For RL, policy iteration methods add noise to the action space, whereby the actions are sampled from an appropriate distribution. As previously mentioned, NES adds noise in the parameter space of the decision variables. For the policy iteration estimator, the variance will grow linearly with the length of the sequence. For NES, the estimator is independent of the sequence length. Since the variance is independent of the time steps simulated for, long-lasting (non-linear) behavioural effects are captured better by NES.

Additionally, black-box optimizers are not sensitive to sampling frequency. Partially due to the gradient estimate being invariant to the length of the episode, which increases robustness to the action frequency. This is mainly due to the missing temporal discounting technique, which is necessary to be applied every time/action step. [96] has also shown the robustness of the hyperparameter settings for NES, which didn't have significant impact on the final optimal policy. In contrast to RL, where a set of hyperparameters must be tuned very precisely for a policy to converge.

Because of the different learning attributes and variable gradient estimation, the reward function for RL and ES should be different too. Use of stochastic policies extracted from long sequences in RL, do not allow the algorithm to converge to sub-optimal control policies first. For ES these sub-optimal strategies are 'reached' quicker than RL, however, have more difficulty completing/converging to the most optimal

strategy, since ES 'finds' every suboptimal control policy along the way. Because of the robuster features black-box optimizers like NES have, and the ability to optimize a policy without knowing the exact internal dependencies, NES could be a useful learning algorithm for SNN's.

8

Towards Neuromorphic Engineering Applications

Neuromorphic engineering is a class of brain-inspired computational methodologies that pursue the understanding and adaptation of the fundamental properties of neural architectures found in mammals [97] [98]. For researchers to be able to implement such technologies, we must transform and modify current algorithms for neuromorphic applications. In order for these algorithms to compute with sparse data and spike based-interactions, processing methods must change. Current algorithms and systems assume or work with continuous data flows. SNN implementations are sparse and discontinuous. This is not only regarded for the computational mechanisms itself, but for the whole system, since inputs (e.g. sensors) and output (e.g. motors) might not adhere to neuromorphic properties.

The authors from [99] were the first to identify and quantify a mathematical model of a neuron into Ordinary Differential Equations (ODE). Since then, many simplifications and adaptations have been made while trying to maintain as much biological resemblance as possible. Simplifying these mathematically defined relationships of neuronal dynamics is necessary for real world deployment and scalability. In order to get the most advantage out of such systems, algorithms and hardware must cohere. The following chapter will first cover the analysis and models for different neuronal elements. After, neuromorphic hardware is discussed. Finally, some robotic applications are presented with a mix of different hardware, learning rules and objectives that make use of the brain inspired computation methodologies.

8.1. Modelling

Before diving into the models that represent different elements of a neuronal system, a brief introduction is given to the most conventional description of a spiking neuron, as described in [100]. A neuron consists of three distinct elements, namely the dendrites, soma and axon, as can be seen in figure 8.1. The soma is the main body of a neuron that performs the key processing step. If the sum of input signals delivered by the dendrites plus the existing potential exceeds a certain threshold, an output signal is generated. The soma sends output signals via the axon that is connected to other neurons. The signals referred to consist of an electrical pulse. An emitted or received pulse does not change its form, which means no information is encoded in the propagation along the axon. Information is only carried by the timing and number of spikes. A sequence of spikes is called a spike train. A pulse travels via the axon to a synapse, which connects an axon and dendrite. In a synapse, a chain of complex biochemical events are triggered when an action potential arrives from an axon. After neurotransmitters have been released out of these events and the transmitter molecules have reached the post-synaptic side, the membrane potential of the receiving neuron will change. The membrane potential is the voltage difference $u(t)$ between the interior of a neuron and it's surroundings.

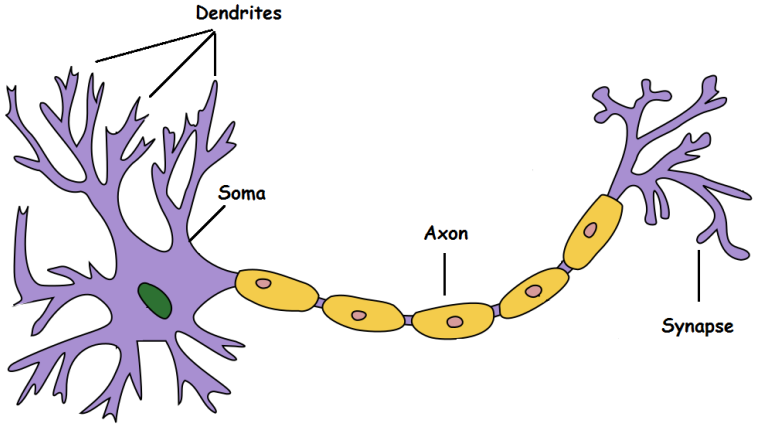


Figure 8.1: Neuronal elements

Positive and negative membrane potential changes are said to be excitatory and inhibitory, respectively. An excitatory synapse will increase the likelihood of an action potential of the post-synaptic neuron, and an inhibitory synapse decreases this possibility. While receiving no input, the neuron is at rest and remains at a constant membrane potential u_{rest}. When the membrane potential has reached a certain firing threshold value, a pulse (action potential) is exhibited that will travel along an axon and synapse to the next neuron. Now that we have covered the basic elements of a neuronal system, the next section will break down synapse, neuron and network modelling properties.

8.1.1. Synapse Models

Several synapse models exist, but differ according to their level of biological resemblance. Although networks consist for a large part of synapses, the main benefits of brain like computations are not accomplished by the inner efficacy of synapses. The processing capabilities, i.e. capturing non-linear behaviour, of bio-inspired neuronal systems are achieved mainly by the neuron's dynamics and network topology. While most use cases require a synapse to attain a simple and static multiplication of a value, the synapse is often the most present and optimized element of a network. Likewise on a physical chip itself, where synapses account for most space filled.

Increasing biological relevance would for many applications significantly increase implementation complexity. Only if modelling biological behaviour is of foremost importance, synapse models would take into account some elaborate form of plasticity or modelling of the chemical interactions such as ion flux [101] and frequency dependent neurotransmitter release models [102](see figure 8.2).

A yet relatively simple and biologically proven phenomenon is spike timing dependent plasticity (STDP). Synaptic plasticity is directly influenced by the exact timing of spikes, as described in [104] [105]. If a pre-synaptic spike has led to a post-synaptic spike, an increase of synaptic strength would be witnessed. Equally, for the reversed order which, in that case, would cause a depression. Neurons that are likely to have contributed to the firing of an action potential of another neuron, are strengthened while inputs that are less likely to have contributed are weakened. The temporal order of pre and post spiking is the most important feature to STDP.

Multiple models have been proposed describing STDP behavior, such as in [100], which is written down below:

$$\Delta t = t_{post} - t_{pre} \tag{8.1}$$

$$STDP(\Delta t) = \begin{cases} A_+ e^{\Delta t/\tau_+}, & \text{if} \Delta t \geq 0 \\ -A_- e^{\Delta t/\tau_-}, & \text{if} \Delta t < 0 \end{cases} \tag{8.2}$$

When $A_+ > 0$ and $A_- < 0$, the synaptic efficacy is increased if the t_{post} spike has fire slightly after t_{pre}. Furthermore, τ is equal to the temporal learning window. Synapse plasticity is a phenomenon that not only

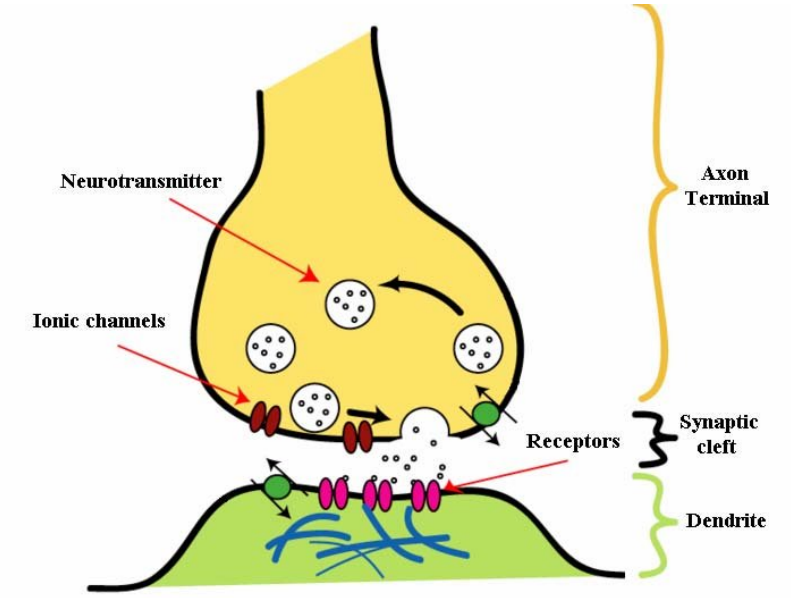


Figure 8.2: The figure is adapted from [103] showing basic attributes of a synapse. Some applications take into account models of ion flow and neurotransmitter frequency dependence.

signifies synapse behaviour, but also has been brought into relation with learning. The author in [106] is the first to theoretically prove that the synaptic plasticity mechanism is used for learning and memory. Also referred to as Hebbian learning, the learning rule that can also be described with the phrase *'Cell's that fire together, wire together with equation'* is shown as,

$$\Delta w_{ij} \propto v_i \cdot v_j, \tag{8.3}$$

where Δw_{ij} refers to a synaptic weight change of a synapse v that is located between pre-synaptic neuron i and post-synaptic neuron j. Equation 8.2 refers to long term potation (LTP) and long term depression (LTD) as mechanisms that underlie cognitive functions as learning and memory. Hebbian learning will be treated further in the next section.

It must be added that there are roughly two types of synaptic plasticity models in terms of an input-output relationship. Firstly, for rate based models, the average firing rate is important. Rate based networks have been a popular approach for converting an ANN into a SNN. In such a manner, ANNs could be trained using conventional learning algorithms (e.g. backpropagation). For spike based networks, more emphasis is put on the exact timing of spikes instead of firing rates in general.

8.1.2. Spiking Neuron Models

As mentioned in the introduction, the authors in [99] were the first to analyse and describe neuronal dynamics into four dimensional non-linear differential equations. As these equations could accurately model biological neural systems, their modelling has been very popular for neuromorphic implementations. Over the years, more models have been created and simplified, but these simplifications have led to a decreasing level of biological plausibility. It is somewhat interesting and self-explanatory that computational applicability is much taken into account. Depending on the task ahead, neuromorphic applications for robotics require large scale networks that have computation time and power constraints, leading to simplified biological models. The following list does not rank the discussed models according to popularity, but to biological level of mimicry.

Although [99] is viewed as the most biologically plausible model, more models do exist that deliver equal level of biological plausibility. Each giving various importance to modelling different neuronal processes. Another biologically plausible model, is the Morris-Lecar model [107]. The model proposes a two-dimensional description of the spike dynamics while maintaining non-linear terms in the differential equations.

The following category of biologically inspired neuronal models do not capture physical activities, but rather try to model behaviour (also called phenomenological models). Additionally, conductance based neuron models are hard to analyse, therefore phenomenological models are preferred for engineering applications. The model of Fitzhugh-Nugamo, as described in [108], reduces the four equations of Hodgkin-Huxley to two equations. This in turn significantly decreases the necessary floating point operations for computations, which is beneficial for computational performance. The model of Izhikevich [109] has a simple but biologically plausible representation that can produce bursting and spiking behaviours. The model accounts for the activation of the K^+ and Na^+ ionic currents that are nested in the membrane recovery variable.

The next category is for Integrate-and-Fire (IF) neurons. Although less complex and less biologically realistic, enough complexity is realized to adhere to spatio-temporal properties. Within the IF category, different levels of complexity exist that model different types of neurons. The most widely used IF model is the Leaky Integrate-and-Fire (LIF) model:

$$\tau_m \frac{\delta u}{\delta t}(t) = u_{rest} - u(t) + R(i_0(t) + \Sigma w_j i_j(t)), \tag{8.4}$$

where τ_m represents the time constant of the membrane modelling the voltage leakage. The effect of the forcing function on the membrane potential u is scaled by R. $i_0(t)$ is an external current influencing the state and $i_j(t)$ is the input current that is multiplied with synapse j weight w_j. If the membrane potential reaches a threshold θ, a single spike is fired, after which the potential u returns to u_{rest}. Often a refractory period follows in which a neuron does not record any activity.

Multiple LIF versions exist. Adaptive LIF adds a mechanism that protects a neuron from firing too extensively by increasing the potential threshold according to the number of action potentials transmitted. Another alternative LIF is the Adaptive Exponential LIF (AELIF) [110], which can produce accurate predictions of a high detailed regular neuron model.

As mentioned previously, models that represent neuronal elements consider computational complexity and biological plausibility. [111] has weighed both attributes by reviewing the necessary floating point operations (addition, division etc.) and biological plausibility in general. Figure 8.3 shows the positioning of neuron models according to biological resemblance and computational complexity of the aforementioned models. It should also be noted that there exists a correlation between floating point operations and the number of parameters present in models. Different kinds and more parameters are generally harder to optimize for, which increases computational effort.

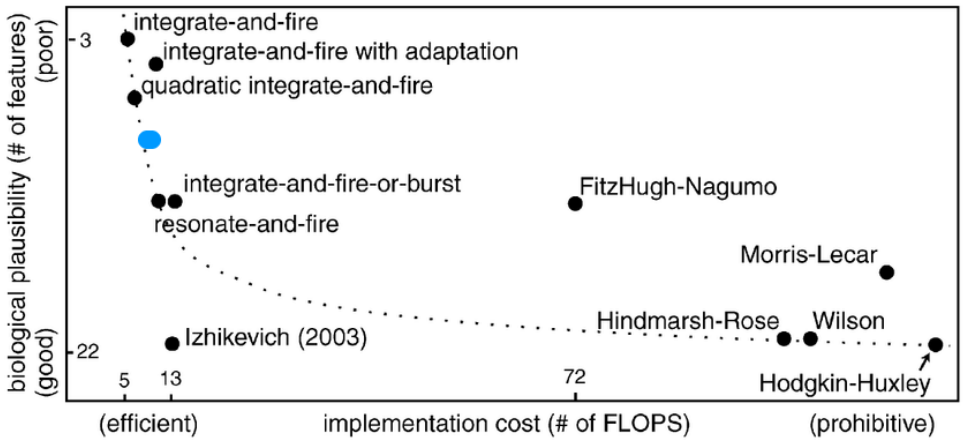


Figure 8.3: Computational effort vs. plausibility. Adapted from [111]. The blue dot depicts the LIF neuron, which is not included in the original image.

The simplifications that have led to the LIF, are simplified around the membrane potential variable. Spike Response Model (SRM) parameters depend on the last time step of firing an output spike. Instead of

differential equations, SRM's are configured using filters that express the membrane potential at a certain time t as an integral over the past [100]. The most important difference is the included refractory period, which is a certain time period in where the possibility of firing spikes is decreased. Regular IF models lack biological plausibility here, as neurons are less likely to fire with equal incitement after the post synaptic spike. This can be done in two ways: 1) Decrease the post synaptic membrane potential, ensuring that input current decays quicker, and/or 2) increase the threshold for a short period of time after firing 8.4. Both methods avoid direct firing to adhere to the refractory property of neurons [112]. This mimics the fact that the ion channels remain more open just after firing, which is why equal incoming input currents have less effect on the membrane potential.

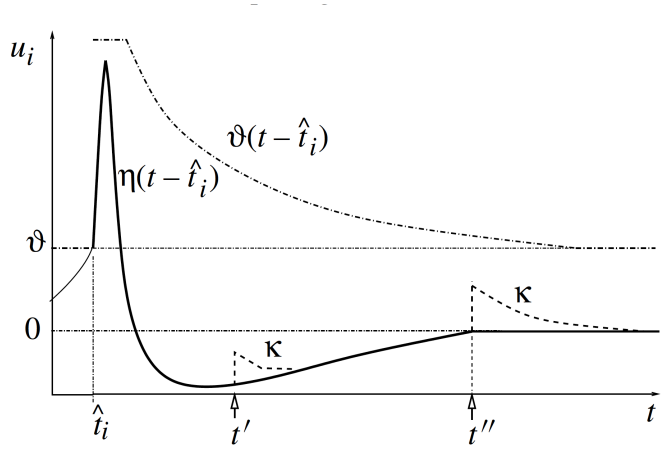


Figure 8.4: Figure adapter from [100]. The figure shows the course of membrane potential $u_i(t)$ as a function of time. The kernel $\eta(t - t_i^t)$ defines the form of the action potential. The figure shows how refractoriness is accomplished by either making the threshold time dependent or the negative after-spike membrane potential. Both influence the threshold for a short period of time. Current k at t' lasts shorter than when applied at t".

8.1.3. Network Models

Networks are populations of neurons that are defined by how synapses are connected. A topology, which defines the structure of a network, models the interactions among neurons. As mentioned in [113], an application's basic requirements can make a certain network type more suitable than others. Furthermore, the grouping of neurons is constrained by the hardware's restrictions of certain topologies. Lastly, the learning algorithm heavily influences the network of choice, since a certain type of synapse connection can not adhere to a training rule. Single layer arrays can also be regarded as a type of network, such as in [114] [115]. Nonetheless, these frameworks do not have the intention of pursuing a biologically feasible application. Since the arithmetic operation dictates the shape of the network, these are not considered in this section.

The most simple division for networks can be attributed to having any connections that propagate impulses 'backwards'. This particular recurrent connection links a neuron with another neuron that is not present further along the intended input to output direction for propagating information.

Recurrent Neural Networks (RNNs) are useful for sequential pattern recognition, since the internal dynamics of such networks allow for capturing historic dependencies from the input. RNN's have feedback connections in order to compute a new dynamical response that is history dependent. Reservoir Computing (RC) is a computational framework that attains a high-dimensional computation space defined by the randomly connected neurons in the reservoir. The high-dimensionality is reached by the large number of spatio-temporal patterns that can exist in the reservoir. The problem is that training an RNN is computationally expensive with backpropagation. Additionally, online recurrent learning is very complex because it is hard to define which error can be attributed to a certain synapse. RC models omit the training of network weights, assuming that the reservoir is able to capture enough and adequate information. The only thing left to do is having to train the interpreting weights in the readout layer that are connected to

random nodes of the reservoir. Two RC models have independently been proposed by [116] Echo State Network (ESN) and [117] Liquid State Machine (LSM). The main difference is that ESN has rate-coded applications and LSM is created to closely mimic the neural microcircuits in brains. The term liquid comes from the probability of a pair of neurons being connected. The closer by these neurons are, the higher chance of a synapse present between them. The echo state property refers to the initial conditions washing out asymptotically as time advances

8.2. Neuromorphic chips

The spatial and temporal calculation properties of neuromorphic processing are much promising in combination with the speed and efficiency traits. Currently, these bio inspired algorithms are adjusted back-end to work on Von Neumann [1] computer architectures. Bio inspired processing and computation software would benefit more from neuromorphic chips that attain the same performance features as designed for with neuromorphic computations. Subsequently, neuromorphic applications would like to make use of hardware that directly supports and mimics biological neural functionalities. Only since the last decade have substantial neuromorphic processors been produced and used.

Loihi chip There are 131072 LIF neurons present on the Loihi chip [97][118] that are partitioned over 128 cores in a spatial and asynchronous grid. Every core possesses memory to keep track of the synaptic state and the routes of the flexibly divisible, 1024 neurons present in a core. The communication between neurons occurs with 32-bit messages that are spike events, which contain the spike properties that are to be shared among the cores. What makes the Loihi chip stand out from other neuromorphic chips is the highly configurable synaptic memory due to its precision in assigning its weight. Additionally, a significant delay of 63 time steps can be applied by the synapse. Next to that, plasticity rules can be applied that change over time according to the spike activity. Three versions of the Loihi chip exist in different configurations, namely the Kapaho Bay, Nahuku and Pohoiki Springs. The latter two are created for larger neuromorphic applications, while the Kapaho bay is practical for small scale operations like MAVs.

TrueNorth The TrueNorth architecture [119] is not only functionally inspired by the brain, but also mimics it's mapping of a column. A cortical column spans from the top to bottom of the brain. The TrueNorth chip consists of 4096 neurosynaptic cores, possesses 1 million neurons and contains 256 million synapses. A single chip can attain a performance of 58 giga-synaptic operations per second and can increase its performance by deploying multiple chips in 4 and 16-chip systems. The TrueNorth chip can attain different neuron models and is able to store neuron connectivity and parameters on locally placed memory. Like the Loihi, the TrueNorth chip allows a pseudo random number generator for different neuron model parameters. For the design, a mix of asynchronous and synchronous elements were chosen, where the communication and control circuits make use of so-called hand-shake protocols. These activate a core only when synaptic inputs are received. The chip itself, which contains all the transistors, is 4.3 cm^2 large and is implemented on the IBM NS1e board as in [120] and [121]. The IBM NS1e board is approximately 15 cm wide and 10 cm long, wherefore an application on a size constrained MAV is less favourable.

Spinnaker The SpiNNaker [122] [123] is an ARM based processor that has been created to support large parallel and bio inspired calculations that possesses a high degree of interconnection. More emphasis has been put on the sorts of communication found in the human brain. The communication infrastructure is optimized to carry numerous of small packets with less memory dedicated to model information. A design compromise has been made for the requirement for deterministic operation, which means that sometimes a packet is dropped to avoid a communication deadlock. Accurate sequential models can be maintained by reimposing deterministic operation, however this evidently is not the most natural way of operating the system. The SpiNNaker is able to run different LIF and Izhikevich models that can hold up to 1000 connected input neurons. Literature has shown that overall, less robotic control implementations of the SpiNNaker chip exist. A robotic application includes the Neuropod [124]. The system makes use of a central pattern generator that generates rhythmic patterns for certain movements. A hexapod robot and a SpiNNaker processor are used to demonstrate the short delays the open-loop control loop acquires when switching between different walking gaits.

8.3. Learning

Due to the non-linear and non-differentiable nature of SNNs, learning performance lags behind the performance achieved by ANNs with conventional learning algorithms. ANNs use differentiable and linear activation functions, which makes it possible for gradient based optimization techniques to adjust connection weights. When using such optimisation, it makes sense to stack layers of neurons (MLP), because the derivatives calculated in one layer are propagated further back in the network. MLPs with many layers are often referred to as Deep Neural Networks (DNN). High computing power enabled the training of DNN's, accomplishing identification tasks better than humans.

A major disadvantage of backpropagation is the lack of biological plausibility and employability on neuromorphic hardware. Brain learning mechanisms allow learning while practising a certain task. The brain can adjust synapse weights real-time, instead of having to wait for some sort of finite evaluation to apply the necessary improvements. The learning rules that allow such real time adaptation are referred to as online rules and provide on-chip parameter adjustments. The learning methods that require a feedback signal every iteration are called offline methods and have much correlation with conventional supervised learning methods. While conventional learning rules mostly are categorized by supervised and unsupervised learning, though, a different division is opted for here. Since biological plausibility introduces more learning aspects to keep in mind, the sections explaining the learning rules, are divided accordingly.

8.3.1. Supervised gradient-based Learning

As highlighted before, backpropagation uses error signals that are propagated through feedback connections to adjust synapses. A gradient can be acquired by minimizing a cost function **L** over a weight:

$$W_{ij} \leftarrow W_{ij} - \eta \Delta W_{ij}, \text{ where} \Delta W_{ij} \frac{\delta \mathbf{L}}{\delta W_{ij}} = \frac{\delta \mathbf{L}}{\delta y_i} \frac{\delta y_i}{\delta a_i} \frac{\delta a_i}{\delta W_{ij}}. \tag{8.5}$$

Here $a = \Sigma_i w_{ij} x_i$ represents the total input to the neuron, y_i the output and η as learning rate. The second term of the last part of the equation tells us which effect parameter changes have on the output of the neuron. For backpropagation gradient descent, this term is calculated for every neuron in the direction of the output prediction neuron to the input synapses.

Although this learning solution can not directly be applied to SNNs, it can be used for SNNs. By first training a CNN with the same topology as the intended SNN will have, the CNN can after training be transformed to a rate-coded SNN. The rates depict a certain magnitude. The previous allows conventional known learning algorithms to be used while still retaining the advantages of neuromorphic processing. [121] showed that a transformed CNN on the TrueNorth [119] chip was able to predict suitable motor outputs.

Deep Learning Networks (DLN) have experienced major learning advancements, especially in capturing long-term dependencies in sequential and temporal data. The combination of training RNNs with the back propagation algorithm have significantly contributed to the achievements in for example speech generation, but also in the continuous control domain [16] for MPC. A key feature of backpropagation that facilitates the robust and accurate weight accreditation applied to the synapses, is due to the stored historical states and outputs over time. However, due to complex non-linear models and environments, it is hard to train a SNN offline (in batches) because there exists a reality gap between the simulation and actual environment. Additionally, offline learning methods require storing and processing states, meaning supplementary memory and processing power is essential, which in turn might harm the scalability of neuromorphic control applications.

As mentioned in the introduction of this section, the non-differentiable nature of spikes makes it hard to calculate a gradient. In order to deal with the discontinuous non-linearities, a surrogate gradient or smoothing function can be introduced. Instead of changing the model, an approximation is used in order to be able to use the proven gradient descent method. However, some adjustments must be made to a in equation 8.5 since the activation function has changed. Smoothing functions allow the network model to be continuously differentiable, while a surrogate gradient operates a continuous relaxation of the real gradients [125]. Surrogate rules can be formulated in a forward and backward sense. The backward method adheres to conventional backpropagation properties, while the forward method could be applied in an online fashion. The latter methods refers to local update rules and is treated in the next section

In [126], Spike Layer Error Reassignment in Time (SLAYER) learning rule is introduced. The authors let the cost function be represented by spike timing and firing rates. This means that the algorithm considers the temporal dependency of pre-synaptic and post-synaptic signals of a spiking neuron. In order to solve the credit assignment problem, the local learning rule calculates a surrogate gradient. As SLAYER is a pure offline learning algorithm, more suitability is pointed towards recognition and classification tasks. The method did produce state-of-the-art results on the N-MNIST data set [127].

8.3.2. Supervised online learning

The credit assignment problem involves the challenge of assigning an appropriate weight to a single synapse instructing a certain strengthening or weakening. The weight transport problem describes the need of having two equal synapses in different directions in order to support back-propagation. Animals do not possess feedback connections of any sort to provide for learning. Thus, conventional learning algorithms are biologically not plausible [128]. Additionally, a major downfall of complex and expensive learning algorithms is that these must be executed offline. An adapted environment is necessary in where external hardware is necessary for calculating appropriate parameters. Our brains, however, are able to accurately adjust parameters while simultaneously executing. Mammal brains are very adaptive. For a similar case regarding offline training, the same scenario must be recalled and trained. Additionally, changing an offline network's topology will have unknown consequences and scales less good than models capable of online learning.

Backpropagation algorithms exist that have taken inspiration from nature and are called Real-Time Recurrent Learning (RTRL) methods. By approximating the same features as used in conventional backpropagation in a feedforward manner, the issue as for online learning is solved. A downside to these algorithms is the large memory necessary to keep track of every synapse's activity while operating.

As will be explained more extensively in the next section, Hebbian learning forms a solid foundation for biologically plausible learning. Hebbian learning is strictly regarded as unsupervised learning, but can be used in combination with other learning signals. Supervised Hebbian learning refers to local modulatory mechanisms being instructed by top-down teaching signals. The eligibility traces (measure of a neuron's activity) and reward signal ensure correct local adjustments in a network. Several algorithms are available that practice these methodologies. SuperSpike [129] is a biologically plausible forward-in-time optimization rule that focusses on aligning target spike trains. Locality refers to low-level activity in neurons to determine their contribution to the output. SuperSpike avoids non-locality by randomly propagating error signals from the output to hidden layers(Learning signal). Another learning rule that falls into this category is e-prop [130]. e-prop has the simplest set-up and only uses eligibility traces and a top-down signal to adjust weights. The learning process takes longer than most other methods, but approaches similar performance. The learning scheme has been compared to others, such as backpropagation, to demonstrate temporal processing capabilities.

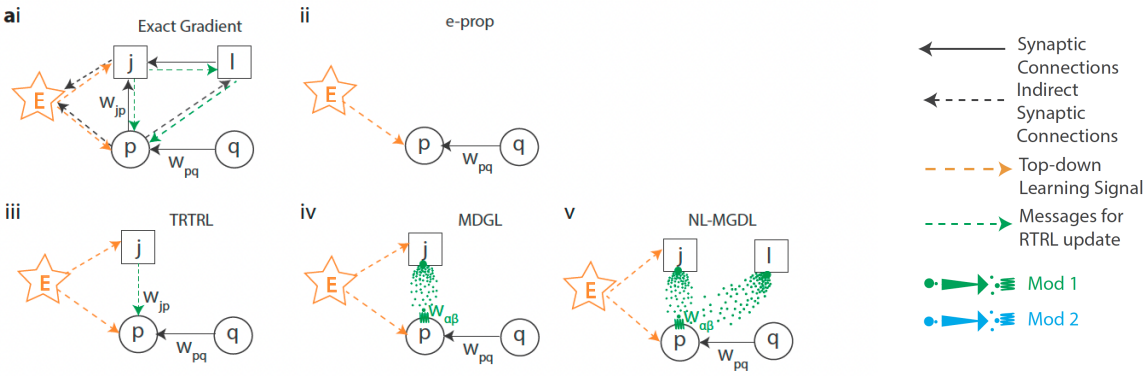


Figure 8.5: Figure adapted from [131] showing cell-type-specific modulatory signals.

Similar to e-prop, the authors in [131] have a top-down learning signal in place, but add a local cell-type-specific modulatory signal. The learning rule is regarded as Multidigraph Learning Rule (MDGL) and lets the local modulatory signal be influenced by neurons around the 'updated' neuron. Figure 7.5 shows the

communication between different cells in a network for learning. 7.5ai shows how backpropagation uses information of other synapses and neurons to calculate weight adjustments. For 7.5iii, the Truncated Real-Time Recurrent Learning (TRTRL) allows dependencies within one connection step. These dependencies are absent for e-prop and replaced by a diffusing modulatory signal for MDGL. Non-Local MDGL (NL-MDGL) (7.5V) includes dependencies that are 'further' located than one connection step.

8.3.3. Unsupervised Learning

Hebbian learning refers to a neuroscientific formulation of a potential causal relation between two firing neurons [106]. A synapse should be strengthened if the pre-synaptic neuron is 'repeatedly or persistently taking part in firing' the post-synaptic neuron. The causality suggested necessitates the pre-synaptic neuron firing slightly before the post-synaptic neuron fires. Spike-Timing Dependent Plasticity (STDP) is a spike-based formulation of Hebbian learning. STDP is a simple concept that links a network's activity to a synapse weight change (learning). Several learning rules fall under the term STDP that base their weight adjustment on the timing between pre- and post-synaptic spikes.

As mentioned before, the firing of two near neurons allows for a learning rule to be formulated. If the pre-synaptic neuron fires just before the post-synaptic neuron fires, the synapse between these neurons is strengthened. A certain temporal threshold Δt determines whether a post-synaptic neuron's firing is due to a pre-synaptic neuron's activity. The strengthening of a synapse is called Long-Term Potentiation (LTP). In the case of a pre-synaptic spike after a post-synaptic firing, it is regarded as Long-Term Depression (LDP). LDP means that repeated spike arrival does not influence any post-synaptic action. If the firing recorded does not fall within a temporal window, any correlation for synaptic adjustments are not recognized. The adjustment of a synaptic weight ω_{ij} is according to:

$$\Delta\omega_{ij} = \begin{cases} Ae^{\frac{-(\Delta t)}{\tau}} & \Delta t \leq 0 \; A > 0 \\ Be^{\frac{-(\Delta t)}{\tau}} & \Delta t > 0 \; B > 0 \end{cases} \tag{8.6}$$

Here, A and B are constant parameters that can vary according to the learning application. τ is the time constant that is equal to the temporal learning window. The time difference between a pre-synaptic and post-synaptic spike is Δt which stands for $t_{pre} - t_{post}$. The learning rule is graphically visualised in figure 8.6.

As [132] explains, networks that have been learned with the STDP rule experience other side effects than only correct output spikes. STDP will make a neuron focus on the synapses of incoming spikes that consistently fire early. The overall response latency of a network decreases by prioritizing signals that propagate fast through a network. This is partially the reason that the brain is good at recognizing patterns. If the input spike train fits a before learned pattern of spikes, the network will output faster.

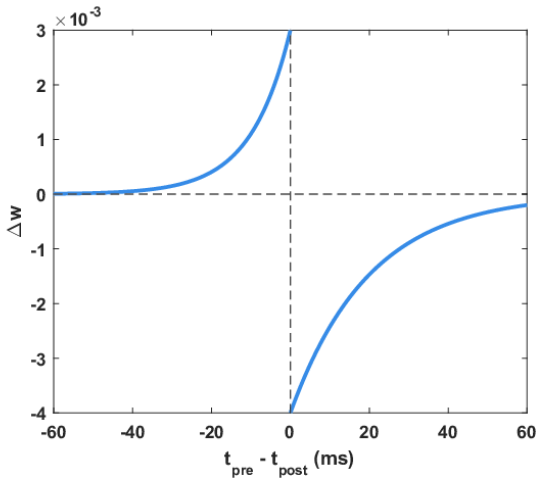


Figure 8.6: Figure adapter from [133]. The figure shows the weight adjustments according to $\Delta t = t_{pre} - t_{post}$. The sequential order of firing will lead to an according weight change.

Although STDP has shown great potential and results already, the learning rule by itself in the simplest form, as shown in 8.6, is biologically not sound and in terms of training, unstable. The already strengthened synapses have a higher probability of strengthening further, while weak synapses will experience LTD. This will result in strong synapses to indefinitely reinforce. This implies that the current weight of synapses also influences future learning.

As discussed before, different forms of the STDP rule exist. In order to solve for the previously mentioned shortcomings, different modifications can be applied. Different kind of mechanisms can be introduced to give the learning process some structure. The parameters that are involved in learning can originate from neuronal activity or self defined user requirements. Two disadvantages arise from introducing complementary mechanisms: 1) More complexity is added to the neuron and/or synapse model and subsequently 2) implementations on hardware are computationally costly.

Additive and multiplicative STDP rules exist that stabilise learning. Additive rules necessitate a hard weight constraint, while most multiplicative rules require some form of weight scaling, rate normalization or some form of activity-dependent scaling. The work in [64] uses a weight scaling solution by implementing a weight-depended exponential rule with pre-synaptic trace information. The learning rule is defined as:

$$\Delta w_{ij} = \eta(LTP + LTD) \tag{8.7}$$

$$\begin{aligned} LTP = LTP_w * LTP_{\hat{X}},\ LTD = LTD_w * LTD_{\hat{X}} \\ LTP_w = e^{-(w_{ij}-\omega_{init})},\ LTD_w = -e^{(w_{ij}-\omega_{init})} \\ LTP_{\hat{X}} = e^{\hat{X}_{ij}} - a,\ LTD_{\hat{X}} = e^{(1-\hat{X}_{ij})} - a \end{aligned} \tag{8.8}$$

where η is the learning rate. The rule is split up in a LTP and LTD contribution of the weight adjustment. Equation 8.8 shows the weight is scaled by the weight at the beginning of the learning step (ω_{init}) and spike activity X_{ij}.

In [134], weights are constrained by setting the total of all weights combined at a pre-determined fixed magnitude. This leads to weight adjustments to certain neurons to be in favour of other neurons so that some neurons will not be significantly larger than others. More examples of adapted STDP learning rules can be found in [135].

As a conclusion of learning algorithms for SNN's, one could conclude the following. The authors in [98] suggest that a paradigm shift is required to understand how the hardware itself can be utilized to explore and optimize training. Until now, learning algorithms have been applied on von Neumann architectures. [98] suggests that a large part of choosing a learning algorithm for SNNs, can unfortunately not yet be made by comparing performance, but leans towards the choice in level of biological plausibility. Therefore, biological plausibility is an important part of the robotic application properties.

8.4. Applications

8.4.1. Encoding

A common decision that must be taken while considering the implementation of SNNs for a continuous control system, is the encoding scheme. Since control systems mostly use a continuous numerical signal, the signal must be converted to a spike-based encoding. The encoding methods influence the possible future intelligence captured from the input. The two most common encoding approaches are rate coding and temporal coding [136]. The encoding schemes will not be discussed elaborately, however, some key observations are given as presented in [137]. First, it is noted that the appropriate encoding scheme mainly depends on the application. This means that the encoding parameters have less effect on the application than the encoding scheme itself. Furthermore, applications achieve higher fitness values if the encoding scheme is more complex. Complexity refers to a combination of encoding schemes or a higher resolution. This might not come too much as a surprise, though it is encouraging to realize that a more complex encoding scheme does not lead to poorer results than a simple scheme. In general, a finer encoded input will lead to a higher objective score. The authors in [138] developed a toolbox in MATLAB and Python, that includes rate, temporal and population encoding schemes. Instead of focussing on the performance, efficiency of the proposed schemes is investigated.

8.4.2. Neuromorphic control methods

Applications of neuromorphic engineering for robotics come in different forms. Some applications focus on transforming well known deep learning methods onto neuromorphic hardware (in order to make use of fast en efficient NE properties above ANNs) and others try to stay as biologically sound as possible regarding adaptation and learning. As [97] already signifies when developing the Loihi chip, a misconception arises that NE is developed to improve current deep learning methods and algorithms. Accordingly, the design of Loihi has favoured creating an understanding of the technology with the according fundamentals theory over accelerating real world commercialized use. It is also important to note that for ANNs designated hardware does not exist. SNNs have the advantage of hardware solutions that fit a solid approximation of dynamics found in mammal like neuronal circuits.

The previous also shines through for neuromorphic control algorithms. Two main categories are identified, namely: 1)a Use NE as framework to (partly) deploy or convert current control and sensing methods and 1)b Build deep learning models and map these to neuromorphic dynamics. 2) Control architectures inspired by insects, often making use of biologically plausible learning rules or applications that fully harness the temporal and non-linear properties of neuromorphic networks.

The following section will highlight the use of neuromorphic processors for different flight control related robotic applications. While we would prefer to dive into neuromorphic landing controllers, there are not plenty of MAV applications. This means other robotic applications are considered in order to complement the overview. Although neuromorphic engineering won't be elaborately explained here, some examples will be given. NE will later be treated in a separate chapter, as it deserves a broader introduction.

As briefly touched upon in the chapter introduction, bio inspired neuromorphic processors have different advantageous properties for control systems. Not only does the framework have a high processing speed and low power usage, the system is able to capture non-linear behaviour of a dynamical system. SNNs consist out of a population of neurons that, according to the values of the neural parameters, are able to output a control signal. The spatio-temporal calculation properties are beneficial for determining a non-linear policy in highly uncertain environments. A subtle distinction can be made in the usage of these neuromorphic processors.

The first category has adjusted the neuromorphic framework for being able to perform arithmetic operations. This means that the neuromorphic architecture is used, ensuring speed and efficiency, but the population topology and decision parameter values are set manually. No learning is involved by interacting with the environment (training or evolving). [6] presents a SNN based PID controller for a one DOF control task of an UAV. The architecture of a neuromorphic chip is utilized to perform arithmetic calculations 8.7. Equally, for [115], though with extended capabilities, improving the controller on the Loihi [118]. [114] is the most recent of the series of research conducted on realizing controllers on neuromorphic hardware that include two separate SNN's for vision and control. Here, the neuronal computing architecture is adapted for traditional algorithms like the Hough transform [139] or PID control to run on. Both utilize neuronal arrays in where the spatial aspect represents a scalar of a conventional algorithm in order for a neuromorphic processor to be used. Additionally, for [114], an adaptation scheme is introduced that compensates for static disturbances 8.8. Neuronal arrays are responsible for converting sparse representations to the familiar PID terms, as presented in [6]. [140] also uses a rate coded neuromorphic implementation that assigns a certain scalar to a position in a neuronal array. The calculated control error is not able to take a range of values, but only the values assigned to in the array.

The second category includes controllers that fully make use of such bio inspired characteristics. The control policy is build by letting the agent interact with its environment in a simulation or either the real-world. The decision variables leading to the most optimal execution of a pre-defined task are used as a control policy. The previous neuromorphic applications had the SNN only to function as a computational architecture. However, when discussing AI based control, the bio inspired calculation properties are used to create a certain behaviour or control policy. While prior applications already know beforehand which neuron parameters they must attain to perform a certain arithmetic calculation, the optimum decision parameters of the second category for a control policy (controller) must be learned. These values are very hard to determine up front, since the exact system dynamics must be known in order to extract the appropriate values of parameters. By simulation or real-world tests, it is tried to find the best decision parameters in controllers to perform a certain control task. The following part will include SNN controllers that constructed a policy fit for the agent and it's environment.

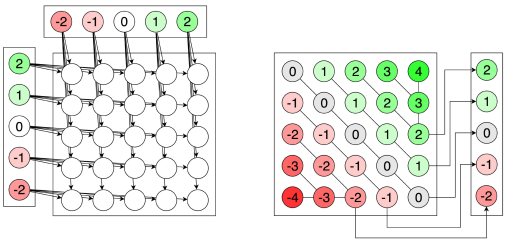


Figure 8.7: This figure is adapted from [6] and shows the arithmetic operations performed on the left. The right array shows possible errors the control signal can attain.

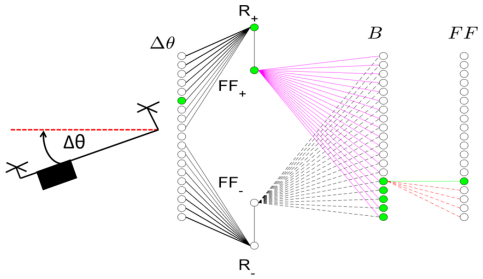


Figure 8.8: The figure is adapted from [114] showing adaptation scheme for minimizing effects caused by constant disturbances

The authors in [141] have evolved SNNs for controlling a pure vertical landing of a MAV. The closed loop controller uses the extracted divergence to control motor thrust for a landing. The algorithm of Lucas-Kanade [36] is used to detect features and subsequently calculate divergence. Research showed that only one neuron established the most preferable landing characteristics, as the projected motor command was able to attain a wide range of thrust commands. The SNNs with a 1 or 20 neurons topology would output only a couple of distinct motor thrusts. In [142], the accuracy performance is shown at which the intended motor commands overlap with the simulated motor commands. A PD controller was implemented to bridge the gap between the SNN set points and actual motor commands.

Another non MAV related robotic application is the robotic arm of [143]. Rate coded populations of neurons represent the inverse kinematics of the robot arm to calculate the necessary motor currents. These motor currents set the joints in the arm to reach a goal position. The neuromorphic application only calculates the error, meaning a secondary controller (PID) is needed to reach the correct states. Other robotic applications that are seen for neuromorphic implementations are the deployment of CNN to SNN by converting traditional NNs to rate coded SNN's. For [121], a CNN was trained to classify images into three classes of motor output: turning left, right and moving forward. After training, the CNN was transformed to run on neuromorphic hardware and was able to successfully navigate an RC car through in- and outdoor environments.

9

Synthesis

The following chapter focusses on the most important findings and theories that are involved in completing the research objective. To highlight relevant sections that answer different parts of the research question, sections are summarized and presented in a conclusive form. The purpose of this study is to collect relevant literature concerning neuromorphic control for biologically inspired and vision-based MAV navigation. Three main areas were identified, namely: 1) Event-based optical flow for biologically inspired landing strategies, 2) Evolutionary frameworks deployable for different kinds of optimization problems and 3) The modelling and challenges involved in neuromorphic computing for robotic applications.

9.1. Biologically inspired vision based navigation

Chapter 4 builds a foundation for the understanding of conventional control properties. The first section 4.1, briefly summarizes available sensing techniques that would suffice for 3D navigation. Next, an overview follows (section (4.2) that analyzes other current control algorithms effective for landings. However, not all of these algorithms are directly useful for visual-based input. An idea is formed of the hardware requirements, operation characteristics and performances. This allows us to compare future neuromorphic control designs to other algorithms. After, intelligent control methods are introduced in section 4.3. These methods use AI to learn the most optimal control behaviour. It is found that machine learning techniques can be deployed for different elements of the control system. Sensing components, trajectory planning and controllers can be trained and have proven to deal well with capturing non-linearities.

To gain insights on vision-based flight control, chapter 5 first discusses OF. Here, estimation techniques are presented that are suitable for frame-based cameras. After listing these methods, the perceived OF is linked to ego-motion by modelling in section 5.2. Derivations and assumptions are discussed that relate flows to biologically inspired visual observables. These features can be used as control input, as is shown in section 5.3, which elaborates on experiments using such strategies. Section 5.4 shows the characteristics of event-based vision, as these are fundamentally different from frame-based perception. The corresponding processing techniques are presented for determining OF with pulse-coded data flow. Furthermore, event-based cameras are considered for different purposes. Following the visual observables, chapter 6 elaborates further upon biologically inspired flight control. By assessing the spatial organisation of the receptive field, an understanding is created of the effects of such organisation. It is shown that some areas have a certain directional sensitivity that help insects determine more accurate OF.

9.2. Optimization frameworks using evolutionary mechanisms

The next chapter essentially analyses why many SNN learning solutions have used optimization schemes based on evolutionary algorithms. Section 7.1 first covers the very basics of EAs. Then, section 7.2 presents popular MOEA, since many practical optimisation problems involve conflicting objectives. It is found that the methods mostly differ in the assessment of separate individuals for future mutations. In order to give future mutations some form of adaption, section **??** introduces ESs. The most well-known ES is CMA-ES. CMA-ES has shown useful properties for evolving systems with discontinuous rewards and in non-linear environments. The algorithm NES (section 7.4) takes it a step further and applies gradient ascent, by means of sampling the evaluation function, to the decision variables. A short comparison to RL is made, since the training and evolution method are both useful for continuous control problems.

9.3. Neuromorphic computing for robotic applications

The last chapter essentially displays the design parameters that must be determined when a neuromorphic application is pursued. These elements consist of models, hardware, encoding and learning properties. First, section 8.1 shows how the basic elements of a neural circuit can be modelled. Synapse, neuron and network models are showed that are based on the neural architectures found in mammals. SNNs, which are regarded as the third generation AI, allow us to simulate the processing characteristics found in the brain systems. Different models are shown that each prioritize different aspects, such as computation complexity and biological plausibility. Section 8.2 discusses the available chips for simulating SNNs. The application constraints dictate the choice of neuromorphic hardware. The next section discusses learning schemes for SNNs. Training SNNs is fundamentally different from ANNs as spikes are non-differentiable. In order to be able to use conventional learning algorithms, adaptions must be applied. Earlier, more effort was put in transforming learning schemes for CNNs to SNNs to benefit from SNN properties, such as speed and efficiency. Other solutions include biologically plausible learning rules that solve for the credit assignment and weight transportation problem. It is evident that online learning algorithms are gaining traction. Not only for closing the real-world gap, but also to maximize the on-chip performance without the need of external hardware to guide the learning process. Furthermore, current advancements where SNNs are implemented in control loops are discussed. The implementations are categorized according to the level of using spatio-temporal properties. It is observed that many applications first implement other types of control algorithms on neuromorphic hardware. Unfortunately, this does not unlock the full processing characteristics of neuromorphic control.

Part III

Closure

10

Conclusion

Due to the limited on board computational resources of MAVs, the processing of visual input and the consecutive navigational decision making require being efficient and fast. Considering the advancements made in event-based processing and neuromorphic hardware, this research investigates the use of SNNs for learning a complex 3D flight control task. A Literature Study is performed which established a reference for the conducted research in the field and the accomplishments regarding integrating neuromorphic hardware in real world robotic applications. Bio inspired feature extraction methods that insects use for navigation during different flight phases were reviewed. Awareness is increasing on how the physical organization and processing characteristics of insects contribute to the cutting-edge flight properties. Inspiration for MAV's is sought in the network architectures, visuomotor coordination principles and control strategies of insects. Furthermore, learning methods were reviewed that showed the challenges linked to learning non-linear and discreet SNNs. A simulation experiment has been established that uses a SNN flight controller for 3D manoeuvres. After consideration of the visual input, it is decided to combine divergence and ventral flow in a single signal. A separate heading error signal was encoded that gives the controller a sense of the sign and magnitude of the heading error. NEAT-SNN was developed, which combines the topology construction mechanism of NEAT and network parameter evolutions for SNNs. The learning algorithm showed that only 80 generations of 50 individuals are necessary to succeed in evolving and custom building a topology for a single network controlling states thrust T, roll ψ and yaw ϕ. Although a single scenario has been trained for whilst optimizing the SNN, this strategy is operable on different flight phases. Simplifying pitch control by adjusting it outside the SNN control system, shows that accurate landings can be performed. The mechanism responsible for pitch adjustment is biologically feasible as it is only using states known by the MAV without external information or sensing necessary. By learning a neuromorphic controller for 3D flight, this work has shown that bio-inspired processing can be used for complex robotic applications.

11

Recommendations

Considering the restricted timeline for the research project, the scope of the project is limited to the topics involved in successfully mapping visual inputs to flight commands. The methodology has discussed multiple assumptions and decisions that, in the light of the Literature Review, might have led to other possible research directions. Equally, the results of the simulation experiment will have uncovered areas where potential gains can be made. The recommendations are split into visual features and the learning algorithm for SNNs.

11.1. Visual feature extraction

While 3D flight control is achieved without controlling pitch θ, further research could focus on including pitch in the output flight control angles. Pitch is not controlled by the SNN in the experiment in the paper, though could be learned by using a different reward function. Since pitch should primarily be dependent on the distance to an object, the objective evaluation could take into account the pitch setting while approaching a target. As such, no outside control loop will be necessary, and every flight control command will be neuromorphically controlled.

As explained in the Literature Review, it is common for vision-based MAV applications to de-rotate the observed flows. This ensures that flows caused by rotation are disentangled from ventral and divergence flows. Many MAV applications use all detected flows in their FOV for processing and motion extraction. Insects, on the other hand, have shown to prioritize the detection of certain motion according to different area's in their FOV. The same motions are detected in different area's of the field of view. Insects might gain additional insights in these differences that result from biases or offsets that are unique to that area. These differences could give the flows additional intelligence about, for instance, absolute distances. An example would be using oscillatory motions to give the MAV a sense of the absolute distance to the ground surface, as shown in [144]. Comparable to angle β in the paper, such variables could help the determination of the MAV's absolute positioning. Future research could focus on which feature to extract in what area to learn additional navigational cues as input for a neuromorphic control system.

11.2. Learning SNNs

The species mechanism of NEAT currently focusses on the differences in synapse properties. The elements that are taken into account are the accumulative weight differences at network level and number of disjoint synapses. NEAT analyses every synapse separately, and therefore might attribute too much learning progress to a single connection. The result would be that all individuals slowly adopt this synapse, which is a successful method for smaller networks. However, if learning progress is a given, it would make more sense for large scale SNNs to, instead, focus on higher order network properties as sparsity and neuron parameters. In general, this research has experienced that the initial network parameters, as initialization weights and neuron parameter values, highly dictate the speed and quality of the learning process. The species mechanism should therefore put more emphasis on higher order network properties than individual network elements. The latter description has much in common with the concept behind HyperNEAT [145]. HyperNEAT searches for optimal connectivity patterns by producing Compositional Pattern Producing Networks (CPPN) that focuses on the spatial aspect of networks. In combination with general EA methods, the quality of learning could be improved by linking the location of synapse or neuron generation to the initialisation values. This could for instance mean that the neurons created nearby the input layer would

have higher decay values to facilitate low pass filtering of visual information. As highlighted in the scientific paper, a part of the learning algorithm consists of synapse generation between two randomly chosen neurons. A large part of successful learning is attributed to synapse generation. Whenever a new synapse is generated, the optimisation that follows starts converging from a new point in the cost landscape. This might help the learning algorithm converge further, instead of being stuck in a 'local minima'. It would make sense to create a connection between two neurons that have potential in improving the performance of the SNN controller. A 'meaningful' connection is created between two neurons when the post-synaptic neuron is in need of information the pre-synaptic neuron provides. By intentionally placing the input and output neurons in a certain order, the encoding neurons that are most likely to be used for mapping information to the output neurons, should be placed closer to each other. By increasing the probability of a synapse generation between neurons that are closer to each other, more 'meaningful' connections will be made. This will result in faster learning, as connections between neurons without correlation are less probable to be created. As seen in the results of the paper, a large part of the network remains inactive. Inactive synapses are the consequence of non-spiking pre-synaptic neurons, whose membrane potential does not reach the threshold value. Instead of solely focusing on synapse creation, synapse pruning could be added to the NEAT-SNN module. This mechanism would get rid of neurons that do not spike and therefore only allow purposeful growth of the network, since all existing neurons would be active. Smaller networks can be built for the same application.

References

[1] John von Neumann. *The Computer and the Brain*. Yale University Press, Dec. 2017. DOI: 10.12987/9780300188080. URL: https://www.degruyter.com/document/doi/10.12987/9780300188080/html (visited on 02/10/2022).

[2] Gordon E Moore. "Cramming more components onto integrated circuits". en. In: 38.8 (1965), p. 6.

[3] Patrick Lichtsteiner et al. "A 128$\times$128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor". en. In: *IEEE Journal of Solid-State Circuits* 43.2 (2008), pp. 566–576. DOI: 10.1109/JSSC.2007.914337. URL: http://ieeexplore.ieee.org/document/4444573/ (visited on 02/16/2022).

[4] Wen Yu et al., eds. *Advances in Computational Intelligence*. en. Vol. 116. Advances in Intelligent and Soft Computing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-642-03156-4. URL: http://link.springer.com/10.1007/978-3-642-03156-4 (visited on 10/06/2021).

[5] Wulfram Gerstner et al. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. en. Cambridge: Cambridge University Press, 2014. DOI: 10.1017/CBO9781107447615. URL: http://ebooks.cambridge.org/ref/id/CBO9781107447615 (visited on 10/06/2021).

[6] Rasmus Stagsted et al. "Towards neuromorphic control: A spiking neural network based PID controller for UAV". en. In: *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, July 2020. DOI: 10.15607/RSS.2020.XVI.074. URL: http://www.roboticsproceedings.org/rss16/p074.pdf (visited on 10/06/2021).

[7] Alvika Gautam et al. "A survey of autonomous landing techniques for UAVs". en. In: *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. Orlando, FL, USA: IEEE, May 2014, pp. 1210–1218. DOI: 10.1109/ICUAS.2014.6842377. URL: http://ieeexplore.ieee.org/document/6842377/ (visited on 07/01/2021).

[8] Tariq Samad et al. "Industry engagement with control research: Perspective and messages". en. In: *Annual Reviews in Control* 49 (2020), pp. 1–14. DOI: 10.1016/j.arcontrol.2020.03.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S1367578820300080 (visited on 09/06/2021).

[9] Atheer L. Salih et al. "Modelling and PID controller design for a quadrotor unmanned air vehicle". en. In: *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*. Cluj-Napoca, Romania: IEEE, May 2010, pp. 1–5. DOI: 10.1109/AQTR.2010.5520914. URL: http://ieeexplore.ieee.org/document/5520914/ (visited on 09/06/2021).

[10] Mohammad Fattahi Sani et al. "Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors". en. In: *2017 International Conference on Computer and Drone Applications (IConDA)*. Kuching: IEEE, Nov. 2017, pp. 102–107. DOI: 10.1109/ICONDA.2017.8270408. URL: http://ieeexplore.ieee.org/document/8270408/ (visited on 09/06/2021).

[11] Mark W. Mueller et al. "Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers". en. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong, China: IEEE, May 2014, pp. 45–52. DOI: 10.1109/ICRA.2014.6906588. URL: http://ieeexplore.ieee.org/document/6906588/ (visited on 09/20/2021).

[12] Davide Falanga et al. "The Foldable Drone: A Morphing Quadrotor That Can Squeeze and Fly". en. In: *IEEE Robotics and Automation Letters* 4.2 (Apr. 2019), pp. 209–216. DOI: 10.1109/LRA.2018.2885575. URL: https://ieeexplore.ieee.org/document/8567932/ (visited on 09/28/2021).

[13] Michael Blösch et al. “Vision based MAV navigation in unknown and unstructured environments”. en. In: *2010 IEEE International Conference on Robotics and Automation*. Anchorage, AK: IEEE, May 2010, pp. 21–28. DOI: 10.1109/ROBOT.2010.5509920. URL: http://ieeexplore.ieee.org/document/5509920/ (visited on 07/15/2021).

[14] Yi Feng et al. “Autonomous Landing of a UAV on a Moving Platform Using Model Predictive Control”. en. In: *Drones* 2.4 (Oct. 2018), p. 34. DOI: 10.3390/drones2040034. URL: http://www.mdpi.com/2504-446X/2/4/34 (visited on 02/10/2022).

[15] Drew Hanover et al. “Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors”. en. In: *arXiv:2109.04210 [cs]* (Sept. 2021). arXiv: 2109.04210. URL: http://arxiv.org/abs/2109.04210 (visited on 09/20/2021).

[16] Benjamin Recht. “A Tour of Reinforcement Learning: The View from Continuous Control”. en. In: *arXiv:1806.09460 [cs, math, stat]* (Nov. 2018). arXiv: 1806.09460. URL: http://arxiv.org/abs/1806.09460 (visited on 10/05/2021).

[17] Grady Williams et al. “Aggressive driving with model predictive path integral control”. en. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. Stockholm: IEEE, May 2016, pp. 1433–1440. DOI: 10.1109/ICRA.2016.7487277. URL: https://ieeexplore.ieee.org/document/7487277/ (visited on 09/21/2021).

[18] Aminurrashid Noordin et al. “Adaptive PID Controller Using Sliding Mode Control Approaches for Quadrotor UAV Attitude and Position Stabilization”. en. In: *Arabian Journal for Science and Engineering* 46.2 (Feb. 2021), pp. 963–981. DOI: 10.1007/s13369-020-04742-w. URL: https://link.springer.com/10.1007/s13369-020-04742-w (visited on 10/01/2021).

[19] Ewoud J. J. Smeur et al. “Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles”. en. In: *Journal of Guidance, Control, and Dynamics* 39.3 (Mar. 2016), pp. 450–461. DOI: 10.2514/1.G001490. URL: https://arc.aiaa.org/doi/10.2514/1.G001490 (visited on 02/10/2022).

[20] Ye Zhou et al. “Extended incremental nonlinear dynamic inversion for optical flow control of micro air vehicles”. en. In: *Aerospace Science and Technology* 116 (Sept. 2021), p. 106889. DOI: 10.1016/j.ast.2021.106889. URL: https://linkinghub.elsevier.com/retrieve/pii/S1270963821003990 (visited on 02/10/2022).

[21] H. W. Ho et al. “Adaptive Gain Control Strategy for Constant Optical Flow Divergence Landing”. en. In: *IEEE Transactions on Robotics* 34.2 (Apr. 2018), pp. 508–516. DOI: 10.1109/TRO.2018.2817418. URL: http://ieeexplore.ieee.org/document/8331925/ (visited on 07/06/2021).

[22] Byoung S. Kim et al. “Nonlinear Flight Control Using Neural Networks”. en. In: *Journal of Guidance, Control, and Dynamics* 20.1 (Jan. 1997), pp. 26–33. DOI: 10.2514/2.4029. URL: https://arc.aiaa.org/doi/10.2514/2.4029 (visited on 12/06/2021).

[23] Elia Kaufmann et al. “Deep Drone Racing: Learning Agile Flight in Dynamic Environments”. en. In: (2018), p. 13.

[24] Yunlong Song et al. “Learning High-Level Policies for Model Predictive Control”. en. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 7629–7636. DOI: 10.1109/IROS45743.2020.9340823. URL: https://ieeexplore.ieee.org/document/9340823/ (visited on 09/20/2021).

[25] S.J. Qin et al. “Nonlinear PLS modeling using neural networks”. en. In: *Computers & Chemical Engineering* 16.4 (Apr. 1992), pp. 379–391. DOI: 10.1016/0098-1354(92)80055-E. URL: https://linkinghub.elsevier.com/retrieve/pii/009813549280055E (visited on 12/07/2021).

[26] Guanya Shi et al. “Neural Lander: Stable Drone Landing Control Using Learned Dynamics”. en. In: *2019 International Conference on Robotics and Automation (ICRA)*. Montreal, QC, Canada: IEEE, May 2019, pp. 9784–9790. DOI: 10.1109/ICRA.2019.8794351. URL: https://ieeexplore.ieee.org/document/8794351/ (visited on 12/05/2021).

[27] Mohammad Jafari et al. “Intelligent Control for Unmanned Aerial Systems with System Uncertainties and Disturbances Using Artificial Neural Network”. en. In: *Drones* 2.3 (Aug. 2018), p. 30. DOI: 10.3390/drones2030030. URL: http://www.mdpi.com/2504-446X/2/3/30 (visited on 12/05/2021).

[28] Fan Jiang et al. “Design, Implementation, and Evaluation of a Neural-Network-Based Quadcopter UAV System”. en. In: *IEEE Transactions on Industrial Electronics* 67.3 (Mar. 2020), pp. 2076–2085. DOI: 10.1109/TIE.2019.2905808. URL: https://ieeexplore.ieee.org/document/8676108/ (visited on 12/05/2021).

[29] J. S. Chahl et al. “Landing Strategies in Honeybees and Applications to Uninhabited Airborne Vehicles”. en. In: *The International Journal of Robotics Research* 23.2 (Feb. 2004), pp. 101–110. DOI: 10.1177/0278364904041320. URL: http://journals.sagepub.com/doi/10.1177/0278364904041320 (visited on 10/20/2021).

[30] Harald E Esch et al. “DISTANCE ESTIMATION BY FORAGING HONEYBEES”. en. In: (1996), p. 8.

[31] Mandyam V. Srinivasan. “Honeybees as a Model for the Study of Visually Guided Flight, Navigation, and Biologically Inspired Robotics”. en. In: *Physiological Reviews* 91.2 (Apr. 2011), pp. 413–460. DOI: 10.1152/physrev.00005.2010. URL: https://www.physiology.org/doi/10.1152/physrev.00005.2010 (visited on 10/20/2021).

[32] Haiyang Chao et al. “A Survey of Optical Flow Techniques for Robotics Navigation Applications”. en. In: *Journal of Intelligent & Robotic Systems* 73.1-4 (Jan. 2014), pp. 361–372. DOI: 10.1007/s10846-013-9923-6. URL: http://link.springer.com/10.1007/s10846-013-9923-6 (visited on 10/20/2021).

[33] Berthold K.P. Horn et al. “Determining optical flow”. en. In: *Artificial Intelligence* 17.1-3 (Aug. 1981), pp. 185–203. DOI: 10.1016/0004-3702(81)90024-2. URL: https://linkinghub.elsevier.com/retrieve/pii/0004370281900242 (visited on 10/26/2021).

[34] E. Rosten et al. “Fusing points and lines for high performance tracking”. en. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*. Beijing, China: IEEE, 2005, 1508–1515 Vol. 2. DOI: 10.1109/ICCV.2005.104. URL: http://ieeexplore.ieee.org/document/1544896/ (visited on 10/24/2021).

[35] Jianbo Shi et al. “Good features to track”. en. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*. Seattle, WA, USA: IEEE Comput. Soc. Press, 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794. URL: http://ieeexplore.ieee.org/document/323794/ (visited on 10/25/2021).

[36] Bruce D Lucas et al. “An Iterative Image Registration Technique with an Application to Stereo Vision”. en. In: (1981), p. 11.

[37] Teahyung Lee et al. “Performance Analysis of A Correlation-Based Optical Flow Algorithm under Noisy Environments”. en. In: *2006 IEEE International Symposium on Circuits and Systems*. Island of Kos, Greece: IEEE, 2006, pp. 4699–4702. DOI: 10.1109/ISCAS.2006.1693679. URL: http://ieeexplore.ieee.org/document/1693679/ (visited on 10/27/2021).

[38] H C Longuet-Higgins et al. “The interpretation of a moving retinal image”. en. In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 208.1173 (July 1980), pp. 385–397. DOI: 10.1098/rspb.1980.0057. URL: https://royalsocietypublishing.org/doi/10.1098/rspb.1980.0057 (visited on 07/07/2021).

[39] Kirk Y.W. Scheper et al. “Evolution of robust high speed optical-flow-based landing for autonomous MAVs”. en. In: *Robotics and Autonomous Systems* 124 (Feb. 2020), p. 103380. DOI: 10.1016/j.robot.2019.103380. URL: https://linkinghub.elsevier.com/retrieve/pii/S0921889019302404 (visited on 07/05/2021).

[40] Davide Scaramuzza et al. “Visual Odometry [Tutorial]”. en. In: *IEEE Robotics & Automation Magazine* 18.4 (Dec. 2011), pp. 80–92. DOI: 10.1109/MRA.2011.943233. URL: https://ieeexplore.ieee.org/document/6096039/ (visited on 10/21/2021).

[41] Luis Rodolfo Garcia Carrillo et al. “Three-dimensional position and velocity regulation of a quad-rotorcraft using optical flow”. en. In: *IEEE Transactions on Aerospace and Electronic Systems* 51.1 (Jan. 2015), pp. 358–371. DOI: 10.1109/TAES.2014.130607. URL: http://ieeexplore.ieee.org/document/7073497/ (visited on 07/06/2021).

[42] Raoul Dinaux et al. “FAITH: Fast iterative half-plane focus of expansion estimation using event-based optic flow”. en. In: *IEEE Robotics and Automation Letters* 6.4 (Oct. 2021). arXiv: 2102.12823, pp. 7627–7634. DOI: 10.1109/LRA.2021.3100153. URL: http://arxiv.org/abs/2102.12823 (visited on 10/25/2021).

[43] Simon Zingg et al. “MAV navigation through indoor corridors using optical flow”. en. In: *2010 IEEE International Conference on Robotics and Automation*. Anchorage, AK: IEEE, May 2010, pp. 3361–3368. DOI: 10.1109/ROBOT.2010.5509777. URL: http://ieeexplore.ieee.org/document/5509777/ (visited on 10/17/2021).

[44] G.C.H.E. de Croon et al. “Optic-Flow Based Slope Estimation for Autonomous Landing”. en. In: *International Journal of Micro Air Vehicles* 5.4 (Dec. 2013), pp. 287–297. DOI: 10.1260/1756-8293.5.4.287. URL: http://journals.sagepub.com/doi/10.1260/1756-8293.5.4.287 (visited on 07/05/2021).

[45] Julien R. Serres et al. “Optic flow-based collision-free strategies: From insects to robots”. en. In: *Arthropod Structure & Development* 46.5 (Sept. 2017), pp. 703–717. DOI: 10.1016/j.asd.2017.06.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S146780391730066X (visited on 11/22/2021).

[46] Axel López et al. “Character navigation in dynamic environments based on optical flow”. en. In: *Computer Graphics Forum* 38.2 (May 2019), pp. 181–192. DOI: 10.1111/cgf.13629. URL: https://onlinelibrary.wiley.com/doi/10.1111/cgf.13629 (visited on 10/26/2021).

[47] Kimberly McGuire et al. “Efficient Optical Flow and Stereo Vision for Velocity Estimation and Obstacle Avoidance on an Autonomous Pocket Drone”. en. In: *IEEE Robotics and Automation Letters* 2.2 (Apr. 2017), pp. 1070–1076. DOI: 10.1109/LRA.2017.2658940. URL: http://ieeexplore.ieee.org/document/7833065/ (visited on 10/20/2021).

[48] Franck Ruffier et al. “Optic Flow Regulation in Unsteady Environments: A Tethered MAV Achieves Terrain Following and Targeted Landing Over a Moving Platform”. en. In: *Journal of Intelligent & Robotic Systems* 79.2 (Aug. 2015), pp. 275–293. DOI: 10.1007/s10846-014-0062-5. URL: http://link.springer.com/10.1007/s10846-014-0062-5 (visited on 02/16/2022).

[49] B. Herissé et al. “Landing a VTOL Unmanned Aerial Vehicle on a Moving Platform Using Optical Flow”. en. In: *IEEE Transactions on Robotics* 28.1 (Feb. 2012), pp. 77–89. DOI: 10.1109/TRO.2011.2163435. URL: http://ieeexplore.ieee.org/document/6017133/ (visited on 02/16/2022).

[50] Alexander Borst et al. “Common circuit design in fly and mammalian motion vision”. en. In: *Nature Neuroscience* 18.8 (Aug. 2015), pp. 1067–1076. DOI: 10.1038/nn.4050. URL: http://www.nature.com/articles/nn.4050 (visited on 11/04/2021).

[51] Bas J. Pijnacker Hordijk et al. “Vertical Landing for Micro Air Vehicles using Event-Based Optical Flow”. en. In: *Journal of Field Robotics* 35.1 (Jan. 2018). arXiv: 1702.00061, pp. 69–90. DOI: 10.1002/rob.21764. URL: http://arxiv.org/abs/1702.00061 (visited on 07/05/2021).

[52] Christoph Posch et al. “An asynchronous time-based image sensor”. en. In: *2008 IEEE International Symposium on Circuits and Systems*. Seattle, WA, USA: IEEE, May 2008, pp. 2130–2133. DOI: 10.1109/ISCAS.2008.4541871. URL: http://ieeexplore.ieee.org/document/4541871/ (visited on 11/05/2021).

[53] Christian Brandli et al. “A 240 × 180 130 dB 3 µs Latency Global Shutter Spatiotemporal Vision Sensor”. en. In: *IEEE Journal of Solid-State Circuits* 49.10 (Oct. 2014), pp. 2333–2341. DOI: 10.1109/JSSC.2014.2342715. URL: https://ieeexplore.ieee.org/document/6889103 (visited on 11/05/2021).

[54] J. Conradt. “On-board real-time optic-flow for miniature event-based vision sensors”. en. In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Zhuhai: IEEE, Dec. 2015,

pp. 1858–1863. DOI: 10.1109/ROBIO.2015.7419043. URL: http://ieeexplore.ieee.org/document/7419043/ (visited on 11/04/2021).

[55] Guillermo Gallego et al. “Event-based Vision: A Survey”. en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020). arXiv: 1904.08405, pp. 1–1. DOI: 10.1109/TPAMI.2020.3008413. URL: http://arxiv.org/abs/1904.08405 (visited on 11/03/2021).

[56] Ryad Benosman et al. “Asynchronous frameless event-based optical flow”. en. In: *Neural Networks* 27 (Mar. 2012), pp. 32–37. DOI: 10.1016/j.neunet.2011.11.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608011002930 (visited on 03/16/2022).

[57] Tobias Brosch et al. “On event-based optical flow detection”. en. In: *Frontiers in Neuroscience* 9 (Apr. 2015). DOI: 10.3389/fnins.2015.00137. URL: http://journal.frontiersin.org/article/10.3389/fnins.2015.00137/abstract (visited on 10/17/2021).

[58] Ryad Benosman et al. “Event-Based Visual Flow”. en. In: *IEEE Transactions on Neural Networks and Learning Systems* 25.2 (Feb. 2014), pp. 407–417. DOI: 10.1109/TNNLS.2013.2273537. URL: http://ieeexplore.ieee.org/document/6589170/ (visited on 03/26/2022).

[59] Bodo Rueckauer et al. “Evaluation of Event-Based Algorithms for Optical Flow with Ground-Truth from Inertial Measurement Sensor”. en. In: *Frontiers in Neuroscience* 10 (Apr. 2016). DOI: 10.3389/fnins.2016.00176. URL: http://journal.frontiersin.org/Article/10.3389/fnins.2016.00176/abstract (visited on 03/24/2022).

[60] Francisco Barranco et al. “Bio-inspired Motion Estimation with Event-Driven Sensors”. en. In: *Advances in Computational Intelligence*. Ed. by Ignacio Rojas et al. Vol. 9094. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 309–321. DOI: 10.1007/978-3-319-19258-1_27. URL: http://link.springer.com/10.1007/978-3-319-19258-1_27 (visited on 03/26/2022).

[61] Alex Zihao Zhu et al. “EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras”. en. In: *Robotics: Science and Systems XIV* (June 2018). arXiv: 1802.06898. DOI: 10.15607/RSS.2018.XIV.062. URL: http://arxiv.org/abs/1802.06898 (visited on 03/27/2022).

[62] Alex Zihao Zhu et al. “Unsupervised Event-Based Learning of Optical Flow, Depth, and Egomotion”. en. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, June 2019, pp. 989–997. DOI: 10.1109/CVPR.2019.00108. URL: https://ieeexplore.ieee.org/document/8953979/ (visited on 10/25/2021).

[63] Nico Messikommer et al. “Event-based Asynchronous Sparse Convolutional Networks”. en. In: *arXiv:2003.09148 [cs, eess]* (July 2020). arXiv: 2003.09148. URL: http://arxiv.org/abs/2003.09148 (visited on 03/26/2022).

[64] Federico Paredes-Vallés et al. “Unsupervised Learning of a Hierarchical Spiking Neural Network for Optical Flow Estimation: From Events to Global Motion Perception”. en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.8 (Aug. 2020). arXiv: 1807.10936, pp. 2051–2064. DOI: 10.1109/TPAMI.2019.2903179. URL: http://arxiv.org/abs/1807.10936 (visited on 10/18/2021).

[65] Garrick Orchard et al. “A spiking neural network architecture for visual motion estimation”. en. In: *2013 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. Rotterdam, Netherlands: IEEE, Oct. 2013, pp. 298–301. DOI: 10.1109/BioCAS.2013.6679698. URL: http://ieeexplore.ieee.org/document/6679698/ (visited on 03/27/2022).

[66] Chankyu Lee et al. “Spike-FlowNet: Event-based Optical Flow Estimation with Energy-Efficient Hybrid Neural Networks”. en. In: *arXiv:2003.06696 [cs]* (Sept. 2020). arXiv: 2003.06696. URL: http://arxiv.org/abs/2003.06696 (visited on 03/26/2022).

[67] Nicolas Franceschini et al. “Optic Flow Based Visual Guidance: From Flying Insects to Miniature Aerial Vehicles”. en. In: *Aerial Vehicles*. Ed. by Thanh Mung. InTech, Jan. 2009. DOI: 10.5772/6491. URL: http://www.intechopen.com/books/aerial_vehicles/optic_flow_based_visual_guidance__from_flying_insects_to_miniature_aerial_vehicles (visited on 11/22/2021).

[68] Ralf Petrowitz et al. “Arrangement of optical axes and spatial resolution in the compound eye of the female blowfly Calliphora”. en. In: *Journal of Comparative Physiology A* 186.7-8 (Aug. 2000), pp. 737–746. DOI: 10.1007/s003590000127. URL: http://link.springer.com/10.1007/s003590000127 (visited on 02/16/2022).

[69] Erich Buchner et al. “Elementary detectors for vertical movement in the visual system of Drosophila”. en. In: *Biological Cybernetics* 31.4 (1978), pp. 235–242. DOI: 10.1007/BF00337095. URL: http://link.springer.com/10.1007/BF00337095 (visited on 02/16/2022).

[70] R Feiler et al. “Ectopic expression of ultraviolet-rhodopsins in the blue photoreceptor cells of Drosophila: visual physiology and photochemistry of transgenic animals”. en. In: *The Journal of Neuroscience* 12.10 (Oct. 1992), pp. 3862–3868. DOI: 10.1523/JNEUROSCI.12-10-03862.1992. URL: https://www.jneurosci.org/lookup/doi/10.1523/JNEUROSCI.12-10-03862.1992 (visited on 02/16/2022).

[71] Simon Laughlin. “A Simple Coding Procedure Enhances a Neuron’s Information Capacity”. In: *Zeitschrift für Naturforschung. Section C: Biosciences* 36 (Nov. 1980), pp. 910–2. DOI: 10.1515/znc-1981-9-1040.

[72] Christian Spalthoff et al. “Neuronal representation of visual motion and orientation in the fly medulla”. en. In: *Frontiers in Neural Circuits* 6 (2012). DOI: 10.3389/fncir.2012.00072. URL: http://journal.frontiersin.org/article/10.3389/fncir.2012.00072/abstract (visited on 11/24/2021).

[73] Karl Farrow. “Lateral Interactions and Receptive Field Structure of Lobula Plate Tangential Cells in the Blowfly”. de. In: (Jan. 2005), p. 114.

[74] Graham K. Taylor et al. “Sensory Systems and Flight Stability: What do Insects Measure and Why?” en. In: *Advances in Insect Physiology*. Vol. 34. Elsevier, 2007, pp. 231–316. DOI: 10.1016/S0065-2806(07)34005-8. URL: https://linkinghub.elsevier.com/retrieve/pii/S0065280607340058 (visited on 11/22/2021).

[75] Klaus Hausen. “Motion sensitive interneurons in the optomotor system of the fly”. en. In: (1982), p. 14.

[76] Jessica Kohn et al. “Eyes Matched to the Prize: The State of Matched Filters in Insect Visual Circuits”. In: *Frontiers in Neural Circuits* 12 (Apr. 2018), p. 26. DOI: 10.3389/fncir.2018.00026.

[77] Holger G. Krapp et al. “Dendritic Structure and Receptive-Field Organization of Optic Flow Processing Interneurons in the Fly”. en. In: *Journal of Neurophysiology* 79.4 (Apr. 1998), pp. 1902–1917. DOI: 10.1152/jn.1998.79.4.1902. URL: https://www.physiology.org/doi/10.1152/jn.1998.79.4.1902 (visited on 11/22/2021).

[78] Holger G. Krapp et al. “Binocular Contributions to Optic Flow Processing in the Fly Visual System”. en. In: *Journal of Neurophysiology* 85.2 (Feb. 2001), pp. 724–734. DOI: 10.1152/jn.2001.85.2.724. URL: https://www.physiology.org/doi/10.1152/jn.2001.85.2.724 (visited on 03/27/2022).

[79] Lance F. Tammero et al. “Spatial organization of visuomotor reflexes in *Drosophila*”. en. In: *Journal of Experimental Biology* 207.1 (Jan. 2004), pp. 113–122. DOI: 10.1242/jeb.00724. URL: https://journals.biologists.com/jeb/article/207/1/113/14753/Spatial-organization-of-visuomotor-reflexes-in (visited on 11/22/2021).

[80] P.J Fleming et al. “Evolutionary algorithms in control systems engineering: a survey”. en. In: *Control Engineering Practice* 10.11 (Nov. 2002), pp. 1223–1241. DOI: 10.1016/S0967-0661(02)00081-3. URL: https://linkinghub.elsevier.com/retrieve/pii/S0967066102000813 (visited on 12/20/2021).

[81] Hwanjo Yu et al. “SVM Tutorial — Classification, Regression and Ranking”. In: *Handbook of Natural Computing*. Ed. by Grzegorz Rozenberg et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 479–506. DOI: 10.1007/978-3-540-92910-9_15. URL: https://doi.org/10.1007/978-3-540-92910-9_15.

[82] Tania Binos. “Evolving Neural Network Architecture and Weights Using An Evolutionary Algorithm”. In: (May 2003).

[83] Matej Črepinšek et al. “Exploration and exploitation in evolutionary algorithms: A survey”. en. In: *ACM Computing Surveys* 45.3 (June 2013), pp. 1–33. DOI: 10.1145/2480741.2480752. URL: https://dl.acm.org/doi/10.1145/2480741.2480752 (visited on 06/23/2021).

[84] K. Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. en. In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002), pp. 182–197. DOI: 10.1109/4235.996017. URL: http://ieeexplore.ieee.org/document/996017/ (visited on 12/20/2021).

[85] N. Srinivas et al. “Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms”. en. In: *Evolutionary Computation* 2.3 (Sept. 1994), pp. 221–248. DOI: 10.1162/evco.1994.2.3.221. URL: https://direct.mit.edu/evco/article/2/3/221-248/1396 (visited on 12/21/2021).

[86] I.K. Nikolos et al. “Evolutionary algorithm based offline/online path planner for uav navigation”. en. In: *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)* 33.6 (Dec. 2003), pp. 898–912. DOI: 10.1109/TSMCB.2002.804370. URL: http://ieeexplore.ieee.org/document/1245266/ (visited on 01/12/2022).

[87] Eckart Zitzler et al. “SPEA2: Improving the Strength Pareto Evolutionary Algorithm”. en. In: (2001), p. 19.

[88] Qingfu Zhang et al. “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition”. en. In: *IEEE Transactions on Evolutionary Computation* 11.6 (Dec. 2007), pp. 712–731. DOI: 10.1109/TEVC.2007.892759. URL: http://ieeexplore.ieee.org/document/4358754/ (visited on 06/24/2021).

[89] Xinqi Zhu et al. “A decomposition-based multi-objective optimization approach considering multiple preferences with robust performance”. en. In: *Applied Soft Computing* 73 (Dec. 2018), pp. 263–282. DOI: 10.1016/j.asoc.2018.08.029. URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494618304897 (visited on 12/20/2021).

[90] Llewyn Salt et al. “Parameter Optimization and Learning in a Spiking Neural Network for UAV Obstacle Avoidance Targeting Neuromorphic Processors”. en. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.9 (Sept. 2020), pp. 3305–3318. DOI: 10.1109/TNNLS.2019.2941506. URL: https://ieeexplore.ieee.org/document/8867860/ (visited on 10/05/2021).

[91] Hans-Georg Beyer et al. “A comprehensive introduction”. en. In: (2004), p. 50.

[92] Nikolaus Hansen et al. “Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)”. en. In: *Evolutionary Computation* 11.1 (Mar. 2003), pp. 1–18. DOI: 10.1162/106365603321828970. URL: https://direct.mit.edu/evco/article/11/1/1-18/1139 (visited on 01/03/2022).

[93] Nikolaus Hansen. “The CMA Evolution Strategy: A Tutorial”. en. In: *arXiv:1604.00772 [cs, stat]* (Apr. 2016). arXiv: 1604.00772. URL: http://arxiv.org/abs/1604.00772 (visited on 01/03/2022).

[94] Daan Wierstra et al. “Natural Evolution Strategies”. en. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. Hong Kong, China: IEEE, June 2008, pp. 3381–3387. DOI: 10.1109/CEC.2008.4631255. URL: http://ieeexplore.ieee.org/document/4631255/ (visited on 06/02/2021).

[95] Tim Salimans et al. “Evolution Strategies as a Scalable Alternative to Reinforcement Learning”. en. In: *arXiv:1703.03864 [cs, stat]* (Sept. 2017). arXiv: 1703.03864. URL: http://arxiv.org/abs/1703.03864 (visited on 10/05/2021).

[96] Felipe Petroski Such et al. “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning”. en. In: *arXiv:1712.06567 [cs]* (Apr. 2018). arXiv: 1712.06567. URL: http://arxiv.org/abs/1712.06567 (visited on 06/16/2021).

[97] Mike Davies et al. “Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook”. en. In: *Proceedings of the IEEE* 109.5 (May 2021), pp. 911–934. DOI: 10.1109/JPROC.2021.3067593. URL: https://ieeexplore.ieee.org/document/9395703/ (visited on 11/05/2021).

[98] Catherine D. Schuman et al. “A Survey of Neuromorphic Computing and Neural Networks in Hardware”. en. In: *arXiv:1705.06963 [cs]* (May 2017). arXiv: 1705.06963. URL: http://arxiv.org/abs/1705.06963 (visited on 11/05/2021).

[99] A. L. Hodgkin et al. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. en. In: *The Journal of Physiology* 117.4 (Aug. 1952), pp. 500–544. DOI: 10.1113/jphysiol.1952.sp004764. URL: https://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004764 (visited on 11/09/2021).

[100] Wulfram Gerstner et al. *SPIKING NEURON MODELS: Single Neurons, Populations, Plasticity*. en. 2002.

[101] Yang Dan et al. “Spike Timing-Dependent Plasticity of Neural Circuits”. en. In: *Neuron* 44.1 (Sept. 2004), pp. 23–30. DOI: 10.1016/j.neuron.2004.09.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0896627304005768 (visited on 11/10/2021).

[102] H. Markram et al. “Differential signaling via the same axon of neocortical pyramidal neurons”. en. In: *Proceedings of the National Academy of Sciences* 95.9 (Apr. 1998), pp. 5323–5328. DOI: 10.1073/pnas.95.9.5323. URL: http://www.pnas.org/cgi/doi/10.1073/pnas.95.9.5323 (visited on 11/10/2021).

[103] Urziceanu Ionut et al. “Design and implementation of a bio-inspired locomotion controller for a differential wheeled robot”. In: (Nov. 2021).

[104] Guo-qiang Bi et al. “Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type”. en. In: *The Journal of Neuroscience* 18.24 (Dec. 1998), pp. 10464–10472. DOI: 10.1523/JNEUROSCI.18-24-10464.1998. URL: https://www.jneurosci.org/lookup/doi/10.1523/JNEUROSCI.18-24-10464.1998 (visited on 11/10/2021).

[105] Henry Markram et al. “Regulation of Synaptic Efficacy by Coincidence of Postsynaptic APs and EPSPs”. en. In: *Science* 275.5297 (Jan. 1997), pp. 213–215. DOI: 10.1126/science.275.5297.213. URL: https://www.science.org/doi/10.1126/science.275.5297.213 (visited on 11/10/2021).

[106] D.O. Hebb. *The Organization of Behavior*. en. 0th ed. Psychology Press, 1949. DOI: 10.4324/9781410612403. URL: https://www.taylorfrancis.com/books/9781135631918 (visited on 11/10/2021).

[107] Alla Borisyuk. “Morris–Lecar Model”. In: *Encyclopedia of Computational Neuroscience*. Ed. by Dieter Jaeger et al. New York, NY: Springer New York, 2015, pp. 1758–1764. DOI: 10.1007/978-1-4614-6675-8_150. URL: https://doi.org/10.1007/978-1-4614-6675-8_150.

[108] William Erik Sherwood. “FitzHugh–Nagumo Model”. en. In: *Encyclopedia of Computational Neuroscience*. Ed. by Dieter Jaeger et al. New York, NY: Springer New York, 2014, pp. 1–11. DOI: 10.1007/978-1-4614-7320-6_147-1. URL: http://link.springer.com/10.1007/978-1-4614-7320-6_147-1 (visited on 11/10/2021).

[109] E.M. Izhikevich. “Simple model of spiking neurons”. en. In: *IEEE Transactions on Neural Networks* 14.6 (Nov. 2003), pp. 1569–1572. DOI: 10.1109/TNN.2003.820440. URL: http://ieeexplore.ieee.org/document/1257420/ (visited on 11/10/2021).

[110] Romain Brette et al. “Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity”. en. In: *Journal of Neurophysiology* 94.5 (Nov. 2005), pp. 3637–3642. DOI: 10.1152/jn.00686.2005. URL: https://www.physiology.org/doi/10.1152/jn.00686.2005 (visited on 11/09/2021).

[111] E.M. Izhikevich. “Which Model to Use for Cortical Spiking Neurons?” en. In: *IEEE Transactions on Neural Networks* 15.5 (Sept. 2004), pp. 1063–1070. DOI: 10.1109/TNN.2004.832719. URL: http://ieeexplore.ieee.org/document/1333071/ (visited on 11/07/2021).

[112] Michael J. Berry et al. “Refractoriness and Neural Precision”. en. In: *The Journal of Neuroscience* 18.6 (Mar. 1998), pp. 2200–2211. DOI: 10.1523/JNEUROSCI.18-06-02200.1998. URL: https:

//www.jneurosci.org/lookup/doi/10.1523/JNEUROSCI.18-06-02200.1998 (visited on 03/01/2022).

[113] Zhenshan Bing et al. “A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks”. en. In: *Frontiers in Neurorobotics* 12 (July 2018), p. 35. DOI: 10.3389/fnbot.2018.00035. URL: https://www.frontiersin.org/article/10.3389/fnbot.2018.00035/full (visited on 11/09/2021).

[114] Antonio Vitale et al. “Event-driven Vision and Control for UAVs on a Neuromorphic Chip”. en. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi’an, China: IEEE, May 2021, pp. 103–109. DOI: 10.1109/ICRA48506.2021.9560881. URL: https://ieeexplore.ieee.org/document/9560881/ (visited on 11/15/2021).

[115] Rasmus Karnoe Stagsted et al. “Event-based PID controller fully realized in neuromorphic hardware: a one DoF study”. en. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 10939–10944. DOI: 10.1109/IROS45743.2020.9340861. URL: https://ieeexplore.ieee.org/document/9340861/ (visited on 12/13/2021).

[116] Herbert Jaeger. “The “echo state” approach to analysing and training recurrent neural networks – with an Erratum note”. en. In: (2001), p. 48.

[117] Wolfgang Maass et al. “Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations”. en. In: *Neural Computation* 14.11 (Nov. 2002), pp. 2531–2560. DOI: 10.1162/089976602760407955. URL: https://direct.mit.edu/neco/article/14/11/2531-2560/6650 (visited on 03/09/2022).

[118] Mike Davies et al. “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”. en. In: *IEEE Micro* 38.1 (Jan. 2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359. URL: http://ieeexplore.ieee.org/document/8259423/ (visited on 10/06/2021).

[119] Filipp Akopyan et al. “TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip”. en. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (Oct. 2015), pp. 1537–1557. DOI: 10.1109/TCAD.2015.2474396. URL: http://ieeexplore.ieee.org/document/7229264/ (visited on 10/06/2021).

[120] Steven K. Esser et al. “Convolutional networks for fast, energy-efficient neuromorphic computing”. en. In: *Proceedings of the National Academy of Sciences* 113.41 (Oct. 2016), pp. 11441–11446. DOI: 10.1073/pnas.1604850113. URL: http://www.pnas.org/lookup/doi/10.1073/pnas.1604850113 (visited on 11/07/2021).

[121] Tiffany Hwu et al. “A self-driving robot using deep convolutional neural networks on neuromorphic hardware”. en. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. Anchorage, AK, USA: IEEE, May 2017, pp. 635–641. DOI: 10.1109/IJCNN.2017.7965912. URL: http://ieeexplore.ieee.org/document/7965912/ (visited on 12/02/2021).

[122] Steve B. Furber et al. “The SpiNNaker Project”. en. In: *Proceedings of the IEEE* 102.5 (May 2014), pp. 652–665. DOI: 10.1109/JPROC.2014.2304638. URL: https://ieeexplore.ieee.org/document/6750072/ (visited on 12/04/2021).

[123] M.M. Khan et al. “SpiNNaker: Mapping neural networks onto a massively-parallel chip multiprocessor”. en. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. Hong Kong, China: IEEE, June 2008, pp. 2849–2856. DOI: 10.1109/IJCNN.2008.4634199. URL: http://ieeexplore.ieee.org/document/4634199/ (visited on 10/06/2021).

[124] Daniel Gutierrez-Galan et al. “Neuropod: A real-time neuromorphic spiking CPG applied to robotics”. en. In: *Neurocomputing* 381 (Mar. 2020), pp. 10–19. DOI: 10.1016/j.neucom.2019.11.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231219315644 (visited on 12/04/2021).

[125] Emre O. Neftci et al. “Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks”. en. In: *IEEE Signal Processing Magazine*

36.6 (Nov. 2019), pp. 51–63. DOI: 10.1109/MSP.2019.2931595. URL: https://ieeexplore.ieee.org/document/8891809/ (visited on 03/07/2022).

[126] Sumit Bam Shrestha et al. “SLAYER: Spike Layer Error Reassignment in Time”. en. In: *arXiv:1810.08646 [cs, stat]* (Sept. 2018). arXiv: 1810.08646. URL: http://arxiv.org/abs/1810.08646 (visited on 03/09/2022).

[127] Garrick Orchard et al. “Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades”. en. In: *Frontiers in Neuroscience* 9 (Nov. 2015). DOI: 10.3389/fnins.2015.00437. URL: http://journal.frontiersin.org/Article/10.3389/fnins.2015.00437/abstract (visited on 03/30/2022).

[128] Timothy P. Lillicrap et al. “Backpropagation and the brain”. en. In: *Nature Reviews Neuroscience* 21.6 (June 2020), pp. 335–346. DOI: 10.1038/s41583-020-0277-3. URL: http://www.nature.com/articles/s41583-020-0277-3 (visited on 03/07/2022).

[129] Friedemann Zenke et al. “SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks”. In: *Neural Computation* 30.6 (June 2018), pp. 1514–1541. DOI: 10.1162/neco_a_01086. URL: https://doi.org/10.1162/neco_a_01086 (visited on 03/09/2022).

[130] Guillaume Bellec et al. “A solution to the learning dilemma for recurrent networks of spiking neurons”. en. In: *Nature Communications* 11.1 (Dec. 2020), p. 3625. DOI: 10.1038/s41467-020-17236-y. URL: http://www.nature.com/articles/s41467-020-17236-y (visited on 03/08/2022).

[131] Yuhan Helena Liu et al. *A solution to temporal credit assignment using cell-type-specific modulatory signals*. en. preprint. Neuroscience, Nov. 2020. DOI: 10.1101/2020.11.22.393504. URL: http://biorxiv.org/lookup/doi/10.1101/2020.11.22.393504 (visited on 03/08/2022).

[132] Amirhossein Tavanaei et al. “Deep Learning in Spiking Neural Networks”. en. In: *Neural Networks* 111 (Mar. 2019). arXiv: 1804.08150, pp. 47–63. DOI: 10.1016/j.neunet.2018.12.002. URL: http://arxiv.org/abs/1804.08150 (visited on 03/08/2022).

[133] Jibin Wu et al. “Competitive STDP-based Feature Representation Learning for Sound Event Classification”. en. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. Budapest, Hungary: IEEE, July 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8851688. URL: https://ieeexplore.ieee.org/document/8851688/ (visited on 03/08/2022).

[134] G. C. Qiao et al. “A Neuromorphic-Hardware Oriented Bio-Plausible Online-Learning Spiking Neural Network Model”. en. In: *IEEE Access* 7 (2019), pp. 71730–71740. DOI: 10.1109/ACCESS.2019.2919163. URL: https://ieeexplore.ieee.org/document/8723038/ (visited on 03/07/2022).

[135] Amar Shrestha et al. “Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning”. en. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. Anchorage, AK, USA: IEEE, May 2017, pp. 1999–2006. DOI: 10.1109/IJCNN.2017.7966096. URL: http://ieeexplore.ieee.org/document/7966096/ (visited on 03/08/2022).

[136] Yang Yi et al. “FPGA based spike-time dependent encoder and reservoir design in neuromorphic computing processors”. en. In: *Microprocessors and Microsystems* 46 (Oct. 2016), pp. 175–183. DOI: 10.1016/j.micpro.2016.03.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S0141933116300060 (visited on 03/30/2022).

[137] Catherine D. Schuman et al. “Non-Traditional Input Encoding Schemes for Spiking Neuromorphic Systems”. en. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. Budapest, Hungary: IEEE, July 2019, pp. 1–10. DOI: 10.1109/IJCNN.2019.8852139. URL: https://ieeexplore.ieee.org/document/8852139/ (visited on 03/16/2022).

[138] Julien Dupeyroux et al. “A toolbox for neuromorphic sensing in robotics”. en. In: *arXiv:2103.02751 [cs]* (Oct. 2021). arXiv: 2103.02751. URL: http://arxiv.org/abs/2103.02751 (visited on 03/16/2022).

[139] J Illingworth et al. “A Survey of the Hough Transform”. en. In: (1988), p. 30.

[140] Sebastian Glatz et al. “Adaptive motor control and learning in a spiking neural network realised on a mixed-signal neuromorphic processor”. en. In: *2019 International Conference on Robotics*

and Automation (ICRA). Montreal, QC, Canada: IEEE, May 2019, pp. 9631–9637. DOI: 10.1109/ICRA.2019.8794145. URL: https://ieeexplore.ieee.org/document/8794145/ (visited on 12/13/2021).

[141] Jesse J. Hagenaars et al. “Evolved Neuromorphic Control for High Speed Divergence-Based Landings of MAVs”. en. In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020), pp. 6239–6246. DOI: 10.1109/LRA.2020.3012129. URL: https://ieeexplore.ieee.org/document/9149674/ (visited on 10/19/2021).

[142] Julien Dupeyroux et al. “Neuromorphic control for optic-flow-based landings of MAVs using the Loihi processor”. en. In: *arXiv:2011.00534 [cs]* (Nov. 2020). arXiv: 2011.00534. URL: http://arxiv.org/abs/2011.00534 (visited on 07/05/2021).

[143] Yuval Zaidel et al. “Neuromorphic NEF-Based Inverse Kinematics and PID Control”. en. In: *Frontiers in Neurorobotics* 15 (Feb. 2021), p. 631159. DOI: 10.3389/fnbot.2021.631159. URL: https://www.frontiersin.org/articles/10.3389/fnbot.2021.631159/full (visited on 06/02/2021).

[144] G. C. H. E. de Croon et al. “Enhancing optical-flow-based control by learning visual appearance cues for flying robots”. en. In: *Nature Machine Intelligence* 3.1 (Jan. 2021), pp. 33–41. DOI: 10.1038/s42256-020-00279-7. URL: http://www.nature.com/articles/s42256-020-00279-7 (visited on 07/05/2021).

[145] Kenneth O Stanley et al. “A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks”. en. In: (2009).