Event-Driven Technologies for Reactive Motion Planning

Neuromorphic Stereo Vision and Robot Path Planning and Their Application on Parallel Hardware

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
(Dr.-Ing.)

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
DISSERTATION

von

Lea Steffen

Datum der mündlichen Prüfung:	27. Juni 2023
Erster Referent:	Prof. Dr.-Ing. Rüdiger Dillmann
Zweiter Referent:	Prof. Dr. sc.nat. Jörg Conradt

Meinem Bruder,
Armando

Acknowledgement

This thesis was conducted during my work as a research scientist at the FZI Forschungszentrum Informatik in the department of Interactive Diagnostic and Service Systems (IDS). Many people have accompanied me on this journey and have positively influenced this work. First and foremost, I want to thank my supervisor, Prof. Rüdiger Dillmann, for giving me the chance to work at this research lab and for his continuous support throughout my thesis. I also want to thank Prof. Jörg Conradt for his interest in my work and for becoming my second supervisor.
I would like to thank our department manager Arne Rönnau for the opportunity to work on many exciting projects, especially NeuroReact and the Human Brain Project. This allowed me to gain important experience and further my education in a wide variety of topics. Sharing them and collecting valuable feedback from colleagues has been a great opportunity and inspiration over the years. Particular thanks are due to Camilo Vasquez Tieck and Jacque Kaiser for introducing me to scientific work during my first years at the FZI as well as Stefan Ulbrich for his consistent dedication to engage in deep scientific discourse. Furthermore, I would like to thank some outstanding students, Robin Koch, Tobias Weyer, Katharina Glück, Daniel Azanov, Max Elfgen, Benedikt Hauck, Rafael Kübler da Silva, Artur Liebert, Philipp Augenstein and Björn Ehrlinspiel. I am also grateful to my IDS colleagues for a great team, the solidarity in stressful times, and for the lively conversations. I would especially like to thank Marvin Große Besselmann and Lennart Puck for their friendship, and a constant source of good humor.
Following a goal over such a long period also requires emotional support. I am therefore deeply grateful to my parents, Claudia and Harald as well as my siblings Mona, Jana, Armando and Madeleine for their unconditional support. Lastly, I want to thank Fabian, for always having my back and for reminding me in the right moments of my strengths.

Abstract (English Version)

Robotics is increasingly becoming a key factor in technological progress. Even though impressive advances have been made, in vision and motion-planning mammal brains still outperform even the most performant machines by far. Industrial robots are very fast and precise, but their planning algorithms do not perform well enough in changing, dynamic environments, as necessary for Human–robot Interaction (HRI). Without adaptive and flexible motion planning, in which the human being is taken into account by the robot, safe HRI cannot be guaranteed. Neuromorphic technologies, including visual sensors and hardware chips, work asynchronously and process spatiotemporal information efficiently. In particular, event-based cameras already outperform their frame-based counterparts in many applications. Hence event-driven methods have great potential to enable faster and more energy efficient algorithms for motion control in HRI. This work presents a method for flexible, reactive motion control of a robotic arm. Thereby the exteroception is achieved by event-based stereo vision, and path planning is implemented in a neural representation of the configuration space. The multi-view 3D reconstruction is evaluated by a qualitative analysis in simulation and transferred to an event-based stereo system. To evaluate the reactive, collision-free online planning a robotic demonstrator was realized and used for a comparative study regarding sampling-based planners. This is complemented by a benchmark of parallel hardware solutions for robotic path planning. The results show that the proposed neural solutions represent an effective way to realize robot control for dynamic scenarios. This work is a basis for neural solutions regarding adaptive manufacturing processes, also in cooperation with humans, without sacrificing speed or safety. Thereby, it paves the way for integration of brain-inspired hardware and algorithms into industrial robotics and HRI.

Abstract (German Version)

Die Robotik wird immer mehr zu einem Schlüsselfaktor des technischen Aufschwungs. Trotz beeindruckender Fortschritte in den letzten Jahrzehnten, übertreffen Gehirne von Säugetieren in den Bereichen Sehen und Bewegungsplanung noch immer selbst die leistungsfähigsten Maschinen. Industrieroboter sind sehr schnell und präzise, aber ihre Planungsalgorithmen sind in hochdynamischen Umgebungen, wie sie für die Mensch-Roboter-Kollaboration (MRK) erforderlich sind, nicht leistungsfähig genug. Ohne schnelle und adaptive Bewegungsplanung kann sichere MRK nicht garantiert werden. Neuromorphe Technologien, einschließlich visueller Sensoren und Hardware-Chips, arbeiten asynchron und verarbeiten so raum-zeitliche Informationen sehr effizient. Insbesondere ereignisbasierte visuelle Sensoren sind konventionellen, synchronen Kameras bei vielen Anwendungen bereits überlegen. Daher haben ereignisbasierte Methoden ein großes Potenzial, schnellere und energieeffizientere Algorithmen zur Bewegungssteuerung in der MRK zu ermöglichen. In dieser Arbeit wird ein Ansatz zur flexiblen reaktiven Bewegungssteuerung eines Roboterarms vorgestellt. Dabei wird die Exterozeption durch ereignisbasiertes Stereosehen erreicht und die Pfadplanung ist in einer neuronalen Repräsentation des Konfigurationsraums implementiert. Die Multiview-3D-Rekonstruktion wird durch eine qualitative Analyse in Simulation evaluiert und auf ein Stereo-System ereignisbasierter Kameras übertragen. Zur Evaluierung der reaktiven kollisionsfreien Online-Planung wird ein Demonstrator mit einem industriellen Roboter genutzt. Dieser wird auch für eine vergleichende Studie zu sample-basierten Planern verwendet. Ergänzt wird dies durch einen Benchmark von parallelen Hardwarelösungen wozu als Testszenario Bahnplanung in der Robotik gewählt wurde. Die Ergebnisse zeigen, dass die vorgeschlagenen neuronalen Lösungen einen effektiven Weg zur Realisierung einer Robotersteuerung für dynamische Szenarien darstellen. Diese Arbeit schafft eine Grundlage für neuronale Lösungen bei adaptiven Fertigungsprozesse, auch in Zusammenarbeit mit dem Menschen, ohne Einbußen bei Geschwindigkeit und Sicherheit. Damit ebnet sie den Weg für die Integration von dem Gehirn nachempfundener Hardware und Algorithmen in die Industrierobotik und MRK.

Contents

Glossary

C_{cont} The boundaries between C_{free} and C_{obst}. ix, 99, 100, 102

C_{free} All configurations in the C-space that can be taken by the robot without causing a collision with obstacles in task space. ix, xi, 99, 100, 102, 104

C_{obst} All configurations in the C-space that would cause a collision because the corresponding part of the task space is occupied by obstacles. ix, xi, 99, 100, 102

canonical neuron Neuron types which are activated while looking as well as grasping an object. ix, 61, 62

disparity Binocular cue to determine depth or distance of an object. ix, 22, 23, 49, 54, 55, 57

end effector Device for handling objects mounted at the end of a robotic arm. ix, xv, 4, 95, 98, 101, 102, 116, 118, 122, 124–127, 129, 179

mirror neuron Neuron types which are activated while grasping an object and observing sombody else grasp an object. ix, 61

space-sweep Monocular technique for depth reconstruction. ix, 51, 52, 57, 177

SPA-cycle Conventional sequence of robot actions in three steps. Sensory perception including data analysis, action planning and blind execution according to plans. ix, 1, 5

task space Subset of the workspace in which the robot operates, thus the space the robot can reach with its end effector. ix, xi, 4, 8, 85, 90, 95, 97–101, 103, 104, 114–116, 118, 122, 124–131, 146, 148, 164, 173, 178, 179

Acronyms

AER Address Event Representation: Hardware protocol for event-based communication. ix, 32, 33, 44, 47, 57, 92, 135, 139, 177

ALU Arithmetic Logic Unit: Combinational digital circuit used in processors. ix, 137

ANN Artificial Neural Network: Network type with analogue neurons and synchronous communication with continuous values. ix, xiv, xvi, xviii, 1, 2, 4, 7, 11, 25, 26, 28, 30, 31, 33, 64, 76, 90, 97, 177

Anti-STDP Anti-Spike-Timing-Dependent-Plasticity: Contrary learning rule to STDP, whereby the activation order of the pre- and postsnaptic neuron is reversed. ix, 29, 30, 146–148, 177

APF Artificial Potential Field: Robotic path planning method that uses attractive forces to reach the goal point and repulsive forces to avoid obstacles. ix, 87, 90

APS Active Pixel Sensor: Semiconductor detector for light measurement. ix, 46–48

ARM Advanced RISC Machines: Microprocessor architecture with low power consumption and high performance. ix, xviii, 33, 134, 135

ASIC Application Specific Integrated Circuit: Electronic wiring realized as an integrated circuit. ix, 33

ATIS Asynchronous Time-Based Image Sensor: Event camera of the manufacturer prophesee. ix, xiv, 9, 44, 46–49, 52, 57, 59, 68, 73–79, 81, 164, 169, 177–179, 181

Batch SOM Batch Self-organizing Map: A deterministic SOM variation. ix, 36, 37, 65

BFS Breadth-first Search: A uninformed search algorithm, which is used to traverse the nodes of a graph. All nodes that can be reached from the current node are considered before descending into depths. ix, xix, 4, 87, 166

BMLS Best Matching Linear Segment: The BMU for SGNG. ix, 113, 114

BMU Best Matching Unit: The neuron that is most similar to the training vector when training a SONN. ix, xiii, 34–40, 42, 62, 68, 71, 73, 106–112, 114, 164

CD Change Detection: Basic technology of all event cameras to generate asynchronous event streams from illumination changes. ix, 46–48, 57

CM C-Measure: Standard metric for SONN representing how well neighborhood structures between neurons are maintained from the input to the output space. ix, 42, 118, 120–123, 181

CNN Convolutional Neural Network: A type of ANN using convolution in place of general matrix multiplication. ix, 58, 60, 69, 142–145, 160

CPU Central Processing Unit: Processor in a computer which executes sequentially instructions as arithmetic, logic, controlling, and input/output operations. ix, 31, 57, 133, 134, 137, 138, 143, 144, 149, 152–162, 176, 179

C-space configuration space: N-dimensional configuration space, where N is the DOF. iii, viii, ix, xi, xvi, 4, 5, 8, 9, 85–87, 90, 91, 94–107, 109, 111, 113–118, 120–124, 126, 128–130, 164, 166, 178, 179

CUDA Compute Unified Device Architecture: Nvidia developed programming interface, which allows program parts to be processed by the GPU. ix, xv, 33, 137, 141, 143, 153, 159, 161, 179

DAVIS Dynamic and Active-pixel Vision Sensor: Event camera of the manufacturer inivation. ix, 46–48, 52, 59, 177

DMP Dynamical Movement Primitives: Extension of MP learning movements by learning from demonstration whereby autonomous nonlinear differential equations are applied to express the motion units. ix, 6, 97, 101

DOF Degrees of freedom: Number of joints in a robot. Each of the n joint angles can be used as a coordinate to span a space in $\mathbb{R}^n$. A full set of joint angles is referred to as configuration q and is sufficient to position every point of a robot in space. ix, xiv, 4–6, 85–87, 89–92, 94–106, 114, 120, 130, 166, 178

DSN Disparity sensitive neuron: Neuron responsible for measuring disparity in cooperative networks for stereo vision. ix, 54, 55

DTRN Dynamic Topology Representing Network: Incrementally growing TRN. ix, 38

DVS Dynamic Vision Sensor: Event camera of the manufacturer inivation and first commercially available representative of this sensor type. ix, 45–49, 59, 177

EM Exposure Measurement: Method of the ATIS to obtain grayscale images with entirely asynchronous image acquisition. ix, 47, 48, 57, 59

EMVS Event-based Multi-View Stereo: Algorithm to produce accurate, semi-dense depth maps from event data. ix, 51, 59

FK Forward Kinematic: Determination of the pose by means of joint angles of the arm elements. ix, 98, 100, 116

FPGA Field Programmable Gate Array: Integrated digital circuit into which a logic circuit can be loaded. ix, 57, 59

FPS Frames per Second. ix

GeNN GPU-enhanced Neuronal Networks: Simulation environment for SNN, based on CUDA technology. ix, 33, 101, 133, 138, 139, 141–145, 149–162, 165, 176, 179, 181

GNG Growing Neural Gas: Extension of a NG that can create and delete neurons during learning to optimally adapt to a data structure. viii, ix, xvi, xvii, 8, 36, 38–41, 99, 107, 110–114, 118–126, 128–131, 164, 171, 173, 178, 181

γ-GNG γ Growing Neural Gas: Variation of the GNG that includes a memory structure with an adjustable depth. ix, 39, 107, 109, 110, 113, 118–122, 130, 171

GPU Graphics Processing Unit: Processor in a computer which is optimized for the calculation of graphics and operates in parallel. ix, xiv, xviii, 7, 8, 31, 33, 57, 64, 133, 134, 136–138, 141, 142, 144, 145, 149–155, 157, 159–162, 165, 179

HH Hodgkin-Huxley: Biologically plausible neuron model for SNN. ix, 27, 28, 92, 94, 95, 141

HPC High Performance Computing: Method of processing large amounts of data at very high speeds using multiple computers and storage devices as a cohesive structure. ix, 134, 140, 142, 143

HRI Human–robot Interaction. iii, ix, 1, 6, 7, 85

IF Integrate-and-Fire: Simple but performant neuron model for SNN. ix, xv, 26–28, 177

IK Inverse Kinematic: Determination of the joint angles of the arm elements based on the poseof the . ix, 4, 85, 90, 95, 98–100, 116

IMU Inertial Measurement Unit: Spatial combination of several inertial sensors such as accelerometers and angular rate sensors. ix, 46

LIF Leaky-Integrate-and-Fire: Extension of IF that incorporates neural leakage. ix, xviii, 26–28, 30, 32, 95, 141, 142, 147, 150, 151

LTD Long Term Depression: Permanent weakening of signal transmission at the synapses of nerve cells. ix, 15, 16, 29, 177

LTP Long Term Potentiation: Permanent strengthening of synapses that leads to a long-lasting increase in neural signal transmission. ix, 15–17, 29, 177

LUT Lookup Table: Array of data that maps input values to output values. ix, 8, 100, 102–104, 114–116, 128, 164, 178

MGNG Merge Growing Neural Gas: Combination of GNG and MNG. ix, 36, 40, 41, 107–113, 118–122, 130, 171, 178

MMM Master Motor Map: Framework for perception, visualization, reproduction, and recognition of human motion. ix, 104, 105

MNG Merge Neural Gas: Extension of a NG that includes temporal context by regarding previous learning progress. ix, xvi, 108, 109, 111, 113, 118, 178

MP Motion Primitive: Brain inspired method for dimension reduction, using synergies between joint movements to adress correlated joints as one unit. ix, xiv, 4, 97, 101

MPCNN Modified Pulse-Coupled Neural Network: Subtype of an SNN. ix, 94, 95

MSOM Merge Self-organizing Map: Extension of the SOM that includes temporal context by regarding previous learning progress. ix, 39, 40, 107–110, 113, 118–122, 130, 171

NEST NEural Simulation Tool, designed for large heterogeneous (spiking) networks of point neurons. ix, xvi, 53, 138–140, 142–145, 149–157, 160–162, 165, 176, 179, 181

NG Neural Gas: Alternative network structure to the SOM, without fixed neighborhood relations and therefore the weight vector is used for adaptation. ix, xv, xvi, xix, 36–38, 40, 41, 107, 108, 110, 111, 113, 118, 131

OMPL Open Motion Planning Library: Open source software for motion planning using sample-based algorithms. ix, 105

PCA Principal Component Analysis: Popular, widely used method for dimensionality reduction. ix, 96, 102

PCL Point Cloud Library: Open source project for 2D/3D image and point cloud processing. ix

PRM Probabilistic Road Map: Robotic planning algorithm that randomly samples the C-space. ix, 4, 6, 8, 89, 117, 128, 129, 164

PSP Postsynaptic Potential: Postsynaptic changes in the membrane potential of a chemical synapse. ix, 14, 15, 26

PyNN Python package for neuronal networks: Designed for simulator independent specification of ANN and especially SNN models. It supports the neural simulator NEURON, NEST and Brian as well as neuromorphic platforms SpiNNaker and BrainScaleS. ix, 53, 139, 140, 142, 144–146, 149, 150, 161, 162

PyNN GeNN PyNN interface for GeNN. ix, 153, 154, 161, 162

QE Quantization Error: Systematic error representing the difference between the continuous input value and its quantized output. ix, 42, 110, 118, 120, 121, 181

RANSAC Random sample consensus: Iteration method for estimating a mathematical model from a data set containing outliers. ix, 69, 77, 79

RDS Random-dot Stereogram: Stereo images of random dots which are used to diagnoses various disorders of binocular vision. ix, 22, 23

RecSOM Recursive Self-organizing Map: Extension of the SOM that includes temporal context by regarding previous learming progress. ix, 39, 108, 109

RL Reinforcement Learning: Umbrella term of methods in which an agent independently learns a strategy by maximizing a reward. ix, 91, 97, 102

RMSE Root-mean-square error: Measure for differences between predicted model values and experimental results. ix, 71, 72, 178

ROS Robot Operating System: Framework for robots also for applications in industrial robotics. ix, 105, 127, 172, 179

RRT Rapidly Exploring Random Tree: Sampling based extension of the A* algorithm which allows the graph to be expanded at any point. ix, xvii, 4, 6, 8, 89, 91, 96, 100, 117, 128, 129, 164

RRT-C Rapidly Exploring Random Tree Connect: Extension of the RRT which achieves a significant increase in speed through building up two tree structures. ix, 89, 105, 128, 129, 164

RSOM Recurrent Self-organizing Map: Extension of the SOM that includes temporal context by regarding previous training samples. ix, 39, 40, 108, 109

RViz 3D visualization tool for ROS. ix, 73, 74, 105, 127

SARK SpiNNaker Application Runtime Kernel. ix, 136

SCAMP SpiNNaker Control And Monitor Program. ix, 136

SDP SpiNNaker datagram packages. ix, 136

SDRAM Synchronous dynamic random-access memory: Semiconductor-based memory variant used as main memory in computers. ix, 135, 136, 141

SGNG Segment Growing Neural Gas: Variation of the GNG which uses segments as basic units. ix, xiii, 40, 107, 113, 114, 118–121, 130, 171, 178

SIFT Scale Invariant Feature Transform: Algorithm for detection and description of local features in images. ix, 58

SIMD Single Instruction Multiple Data: type of parallel processing. ix, 137, 138

SLAM Simultaneous Localization and Mapping: Process in which a mobile robot simultaneously creates a map of its environment and estimates its spatial position within this map. ix, 59, 91

SM Streaming Multiprocessor: General purpose processors with a low clock rate target and a small cache, used in GPU architectures. ix, 137

SNN Spiking Neural Network: Type of ANN which is distinctly closer to an actual brain and well auited to deal with time related data, as their neurons communicate asynchronously with spikes. viii, ix, xv, xvi, xviii, 1, 2, 4–9, 25, 28–31, 33, 38, 52, 59, 67, 69, 91, 92, 94, 95, 101, 117, 131, 133–136, 138–141, 143–146, 148–150, 160, 162–166, 175, 179, 181

SOM Self-Organizing Map: Best-known candidate of the SONN, has a rigid neighborhood structure. vii, ix, xvi–xix, 4, 9, 33–41, 58, 60–73, 76–82, 97–99, 102, 107–110, 113, 114, 118–124, 126, 130, 131, 164, 166, 171, 173, 177, 178, 181

γ-SOM γ Self-organizing Map: Variation of the SOM that includes a memory structure with an adjustable depth. viii, ix, 39, 107, 109, 110, 113, 118–122, 124–126, 130, 171, 178

SOMSD SOM for structured data: Extension of the SOM that allows the mapping of structured objects into a topological map using unsupervised learning. ix, 39, 108

SONN Self-organizing Neural Network: Unsupervised learning method producing a low-dimensional representation of the input data while preserving its topological properties. viii, ix, xiii, xiv, xviii, 8, 9, 33, 34, 36, 38–42, 61, 97, 102–108, 110, 111, 113–115, 117–123, 125, 126, 128–130, 163, 165, 171, 179, 181

SOS Self-Organizing System: Brain structures, already present before birth, that apply self-organized processes. ix, 17

SpiN1API SpiNNaker1 API. ix, 136, 140

SpiNNaker Spiking neural network architecture: A representative of neuromorphic hardware, designed to support large scale simulations of SNN, built of ARM microprocessors. ix, xvi, 32, 33, 57, 95, 101, 133–136, 138–140, 142–145, 149–162, 165, 176, 179, 181

SRAM Static Random Access Memory: Electronic volatile memory device. ix, 135

SRM Spike Response Model: Neuron model for SNN that is closely related to LIF. ix, 27, 95

STDP Spike-Timing-Dependent-Plasticity: Synaptic learning rule in which weight update depends on the temporal correlation of spikes between the pre- and post-synaptic neuron. It has been observed in the brain and is commonly used for training SNN. ix, xiii, xix, 2, 8, 29–31, 92, 93, 95, 140, 146, 148, 151, 152, 157, 160, 165, 177

SVF Synaptic Vector Field: Result of synaptic strength changes caused by STDP in the course of a neural WFA. ix, 147, 148, 151, 157, 159, 161, 162, 179

SVM Support Vector Machine: Supervised learning method for classification and regression analysis. ix, 100

TKM Temporal Kohonen Map: Biologically plausible variation of the SOM for temporal data analysis. ix, 39, 40, 108, 109

TRN Topology Representing Network: Combination of NG and a competitive Hebbian learning rule to define a synaptic structure. ix, xiv, xix, 36, 38, 111

UDP User Datagram Protocol: Communication protocol for low-latency, loss-tolerant connections. ix, 136

UR Universal Robots: Danish manufacturer of industrial, collaborative lightweight robots. ix, 106, 114–116, 127, 128

URDF Unified Robot Description Format: XML format for representing a robot model. ix, 66

ViSOM Visualization Induced Self-Organizing Map: Extension of the SOM that preserves inter-point distances allowing an effective visualization of data structure and distribution. ix, 36, 40

VLSI Very Large Scale Integration: Complex digital circuits with several hundred thousand transistors and up to several billion transistors. ix, 31

WFA Wavefront Algorithm: Path planner in a discretized robotic workspace using a BFS on the graph induced by the neighborhood connectivity. viii, ix, xix, 4, 5, 9, 86–89, 91–93, 95, 100, 101, 117, 123, 124, 131, 145–152, 154, 157, 159, 161, 164–166, 178, 179, 181

WINN Weighted Incremental Representing Network: Subtyp of TRN. ix, 38

1. Introduction

Industrial robots are fast, strong and precise, which makes them ideally suited for automation processes in production lines. A historically significant control architecture is the Sense-Plan-Act-cycle (SPA-cycle). This paradigm uses sensor data to create a world model, on which planning is performed. The plan is subsequently executed without any direct link to the sensors [1]. Popular in the 80s, it is considered bad design because all complexity is only in one module, the *plan* part of the system. Good software design, however, encapsulates complexity in equal amounts [2]. The SPA-cycle is only useful in case of a static environment and a slow sensing process [2; 1]. However, as the need for product customization has steadily increased in recent years and different types of tasks can be better performed by humans and others by robots, application areas for Human–robot Interaction (HRI) have been expanding. Collaborative workstations require flexible and reactive robotic motion control to ensure human safety.

1.1. Motivation

As humans handle collaborative tasks with ease, the use of biologically inspired technologies and methods is very promising. The brain processes information sparsely and asynchronously by the use of neural activation pulses, referred to as spikes. Spike trains, time-wise patterns of these fast depolarization impulses, convey information about all sensory input. The information content is decoded in the sequences of identical spikes and neural computation means to process these spike trains [3]. The mechanics of the human brain serve as a model for Artificial Neural Networks (ANNs), which are applied to versatile machine learning tasks. ANNs consist of analog neurons which are based on a differentiable activation function. An exemplary visualization of the intended use case is given in Figure 1.1. Hereby, a robot and a human are shown in a shared workspace. It visualizes which information is necessary to enable motion planning of the robot and how this is represented in the case of a neuronal solution.

However, the source of the brain's enormous flexibility, speed, power efficiency and fault tolerance is spike-based communication. Inspired by this observation, Spiking Neural Network (SNN) embody neurons with a membrane potential that evolves in time depending on the input of weighted spikes. As spiking neurons, unlike analog neurons, consider temporal dynamics, by encoding temporal information in their signals, they are in theory more powerful. Additionally SNN emphasize the neurobiological aspects of neurons, making them distinctly closer

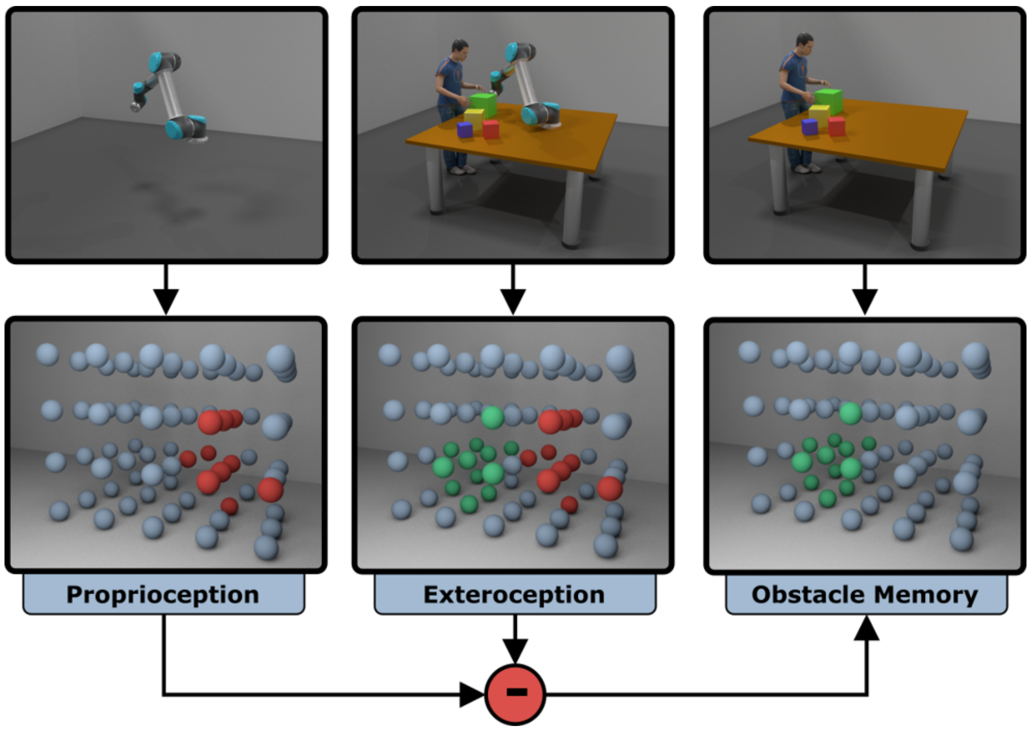


Figure 1.1.: Simulated and neuronal representation of an exemplary realization of the intended use case. The upper row visualizes a scene, perceived on the left by the proprioception of the robot and in the middle by exteroception by an external vision component. The obstacle-free workspace, thus the space in which the robot can move freely, is shown on the right. How this could be represented in the neural space is visualized in the lower row. Thereby, gray nodes represent unoccupied space, red ones the robot and green ones the obstacles. Image source: (Steffen et al. 2019a; Hauck 2019)

to an actual brain than their predecessor [4; 5; 6]. SNN, specially designed to deal with time-related data, embody more complex structures than the clean layers of ANNs, also known as state-of-the-art deep learning networks. Thus, SNN demand more complex learning algorithms, which, up until now, are predominantly based on Spike-Timing-Dependent-Plasticity (STDP). The learning rule STDP strengthens synapses locally in case of correlated activities and was originally formulated in [7; 8]. To exploit the benefits of spike-based computation, its application on parallel hardware, mostly but not mandatory neuromorphic chips, is needed [9; 10; 11; 12; 13; 14]. As event-based sensors, also referred to as silicon retinas, utilize asynchronous and sparse communication like SNN, there are interesting synergies. These bio-inspired models differ from conventional frame-based cameras in their manner and frequency of image acquisition and data transfer. The conventional way of imaging the entire frame at a given rate is replaced by an independent per-pixel response to illumination changes in event-based sensors. As a result of this procedure time location and polarity, the sign of change in brightness is encoded in the output event-stream [15]. The advantages to being derived therefrom are energy efficiency, low redundancy and the simultaneous realization of the contradictory features of high-speed processing and high temporal resolution. The advantages resulting from these properties are significant for robotics and computer vision. However, as a system for path planning is heavily dependent on a 3D representation of the environment and imaging is in its nature planar, depth reconstruction is necessary. A popular method is stereo vision, which obtains depth by matching several perspectives and computing the disparities. While the interaction of both eyes and the brain seems to compute disparities effortlessly, artificial techniques imply high latency and power consumption. The root of the problem is to find the corresponding points in perspectively deviating images of a scene, referred to as the correspondence problem. Due to offering a new physical constraint –time– matching candidates can be drastically reduced (Steffen et al. 2019c). Unfortunately, previous research

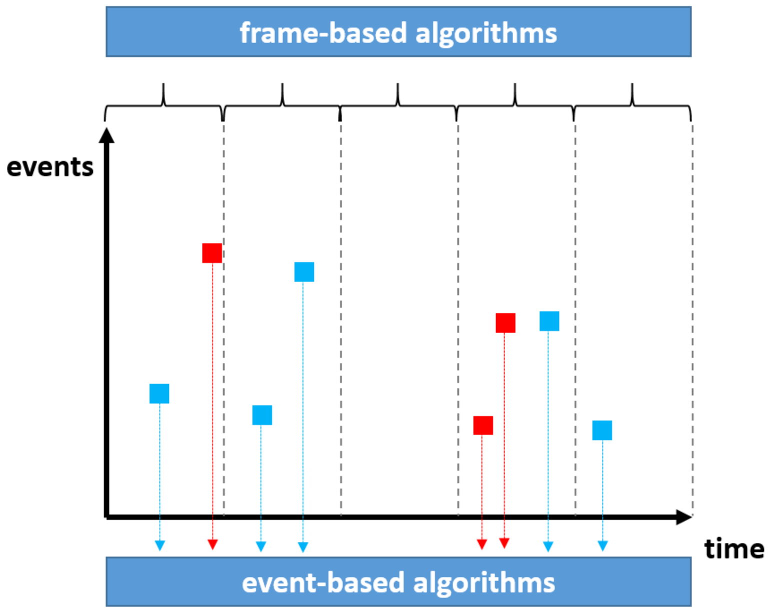


Figure 1.2.: Structural differences of frame-based and event-based vision algorithms applied to asynchronous event streams. The former concept integrates events in a certain time interval, discarding the precise timing information. The latter exploits data more sophistically, by using each event's exact timestamp. Graphic is inspired by [16]

in computer vision and its application in robotics is frame-based. This concept does not exist for event-based sensors. Therefore, most conventional algorithms, also about stereo reconstruction, cannot be used in this way. As visualized in Figure 1.2 there are different approaches to deal with this problem. The obvious but not very elegant method is to convert event streams into frames allowing the use of many performant, matured algorithms. However, the advantages of bio-inspired vision cannot be exploited in this way. Alternatively, new event-based algorithms must be found. A particularly good approach is to use SNN as they are perfectly suited for event-based input. Furthermore, the combination of these technologies results in promising synergies, based on their biological plausibility. Robot motions can be either planned in the 3D Cartesian task space T or the higher dimensional configuration space (C-space) of robots. For path planning, most methods plan a trajectory in the Cartesian space and transfer them to the C-space with Inverse Kinematic (IK) [17; 18]. Unfortunately, that may lead to redundancies, as several solutions in the C-space are applicable for one point in the task space. Planning in the C-space is generally more powerful than in the task space, as it takes into account not only the end effector but all joints and therefore also the robot's "elbow". Consequently, arm movements are smoother and well-validated, meaning no inherent collisions or sudden large changes in joint angles. Although, as a rising number of Degree of freedom (DOF) increases the dimensionality of the search space, for robots with advanced kinematics, planning in the C-space becomes disproportionately more complicated and computationally expensive. Traditionally, grid-based methods are often used for path planners in 2D as they are optimal and complete but do not scale well with higher dimensions. Motion planning in a high dimensional C-space is usually achieved by sampling-based planners like Probabilistic Road Map (PRM) [19] or Rapidly Exploring Random Tree (RRT) [20]. These techniques are more efficient but do not always provide optimal solutions. The relevant literature knows several methods to reduce the search space either with Self-Organizing Map (SOM) or Motion Primitives (MPs) [21; 22]. The biologically inspired MPs reduces the dimensionality by applying only a reduced set of adjustable MPs, commonly embodied by splines or ANNs. Furthermore, the Wavefront Algorithm (WFA) which consists of a Breadth-first Search (BFS) [23] is an effective method for path planning with discrete grid maps [24]. On the other hand, the brain solves motion control through extreme parallelization [25; 26]. Hence, for neural path planning in the C-space, discrete configurations can be represented by neurons and synapses are consequently used for path planning. Due to its biological plausibility and exploitation of parallel structures, this use case is well suited to be applied to neuromorphic systems [27]. As precise motion planning requires a high resolution of the C-space many neurons and synapses are necessary. Unfortunately, network computations are drastically slowed down by large amounts of neurons and dedicated hardware only provides a limited amount of neurons. Consequently reducing the search space through pruning superfluous neurons and synapses is required.

1.2. Research Goal and Problem Statement

> **Research goal 1.** This work aims to develop a holistic neural system for motion control of a 6 DOF robotic arm. The SPA-cycle which is not suited for fast and dynamically changing environments, is to be replaced. As visualized in Figure 1.3, exteroception is generated by event-based stereo vision, combined with the internal robot state. Thereby a neural representation of the workspace is obtained which can be further transferred into a representation of the C-space. Path planning is subsequently performed via the WFA on neuromorphic hardware.

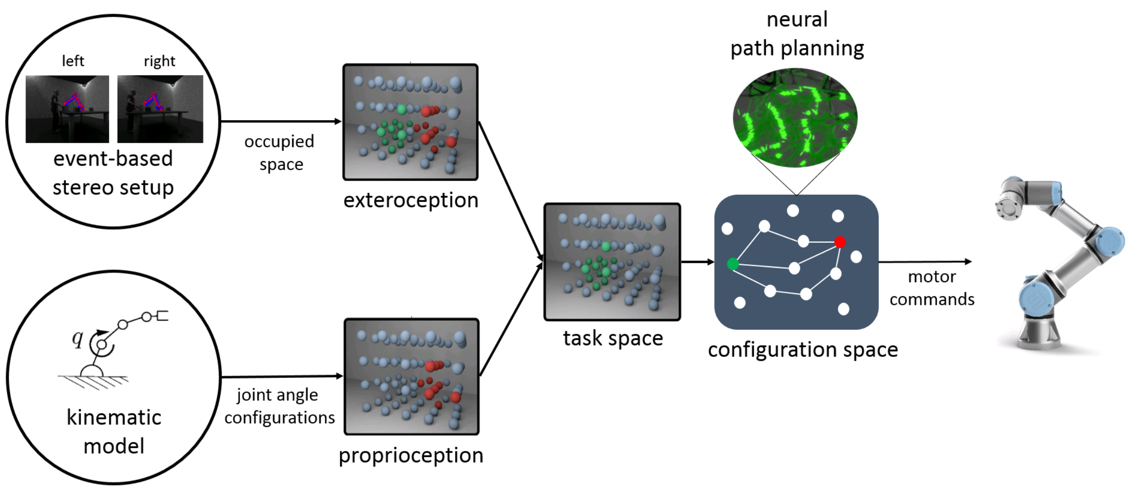


Figure 1.3.: Conceptual architecture of the proposed approach.

Several problems can be derived from these goals and are defined by the following research questions:

> **Research question 1.** How can asynchronous event streams be optimally exploited for event-based stereo vision?

> **Research question 2.** Is it feasible to use the high-dimensional C-space, which requires huge amounts of neurons, for neural path planning?

> **Research question 3.** How can parallel hardware help to exploit the advantages of SNN?

1.3. Thematic Classification and Scientific Contribution

This is a biologically inspired thesis, thus, the underlying concepts are often based on mechanisms observed in nature. With that in mind, it must be emphasized that this work does not try desperately to maintain biological plausibility. On the contrary, attempts were made to implement brain-inspired techniques for industrial applications. This applies in particular to robotic motion planning for HRI.

State-of-the-art The generation of collision-free motions for robots has always been a fundamental challenge. Accordingly, the literature regarding reactive planning in a dynamic environment including collision detection is extensive. Complete and optimal methods guarantee finding the shortest path if one exists [28; 29], even regarding dynamic obstacles [30]. However, these techniques are computationally complex and even their modern extensions [31; 32], struggle with many DOF. On the other hand, sampling based algorithms, like PRM [19], RRT [20] and especially their extensions [33; 34], are very fast and efficient. However, these algorithms are not deterministic. This does not only mean that the generated path is not necessarily the shortest one, but that different paths are found if executed multiple times under identical conditions. So far there have only been a few convincing approaches to path planning in high dimensions. An example that is already widely used in practice, is Dynamical Movement Primitives (DMP) [35]. This method was explicitly developed to learn from demonstration, limiting their generalization. The original version, for example, is not able to derive common behavior from multiple observations, although this problem was recently addressed in [36]. Regarding neural algorithms, especially with SNN, there is very little research for path planning in 3D. Up until recently methods for path planning using SNN have been predominantly applied to 2D [37; 38; 39]. Many of them are based on research about spatial perception and navigation in nature [40; 41]. In particular, the discovery of place cells [42] has inspired many respective approaches [38; 39; 43; 44]. These methods represent interesting candidates for generating robotic motion if extended to 3D, as done in [45; 46], particularly when applied to neuromorphic hardware [47; 48; 49].
To make path planning possible, even in a constantly changing environment, a type of 3D sensing is required. 3D information can be obtained in multiple ways. On the one hand in an active manner, like radar, LIDAR, ultrasonic sensors, light section and structured light. On the other hand passive methods like structure from motion, shape from shading and stereopsis. These methods are mostly either slow, very resource and respectively computationally intensive or have an extremely high information output that is difficult to process further. For applications such as navigation and robotic motion control, therefore predominantly cameras are used, as they generate dense data in real-time. However, two issues exist regarding cameras. Firstly, frame-based visual sensors generate highly

redundant data whose temporal resolution cannot capture fast movements. Consequently, simultaneous over and under-sampling is the result. Secondly, since cameras only generate 2D data, information about the depth is lost. The development of event-based visual sensors [15] enables the circumvention of the first problem. The second issue can be tackled by fusing image data from two, or more, slightly shifted perspectives. This has been intensively investigated [50; 51], as a large proportion of research in the field of computer vision deals with 3D reconstruction. However, this research solely focuses on frame-based cameras, and as visualized in Figure 1.2, it is not purposeful to shift respective techniques to event-based data. Hence, there is still a gap in the state-of-the-art regarding the 3D reconstruction of event-based data. A few methods have already been presented. The best known are cooperative algorithms [52; 53; 54; 55; Kaiser et al. 2018], implementing the basic research about binocular vision by Poggio and Marr [56]. While very strong in terms of biological plausibility, these methods are often limited in terms of practical applicability. Convincing approaches, that exploit the full potential of event cameras, are still missing. In addition, there are no methods to evaluate the existing techniques.
Although SNN are theoretically far superior to conventional ANN, their applications are still often outperformed by methods based on deep learning. There is an urgent need for hardware/software co-design, regarding SNN. The issue that SNN simulation is very slow on conventional hardware can be circumvented by parallel hardware solutions. A plenitude of neuromorphic chips and neurosimulators exist, but, many of them are designed for specific use cases. There are only a few comprehensive benchmarks for these systems. Also, many of these relate to neuroscience scenarios [57; 58] or vision [59; 60]. There is very little research that relates to a robotic use case, and that available [61; 62], neglects Graphics Processing Unit (GPU)-based SNN simulation.

Contribution This thesis approaches the issue of reactive and flexible path planning, which is necessary to allow HRI, from various directions. Its primary scientific contributions are:

- A comprehensive literature survey, for each of the three topics resulting from the research questions in section 1.2. Of particularly large impact is the overview of event-based depth reconstruction, an extensive collection regarding the historic development as well as the latest achievements of respective algorithms.

- Co-development on a new biologically inspired method for stereo reconstruction which is very well suited for event-based data. A special achievement of the approach is that it is not strongly dependent on calibration.

- Further development and transfer of path planning algorithms with SNN from 2D to 3D, based on two different models.

- A biologically inspired novel concept for high-reactive path planning in real-time. The dimensionality of a robot's C-space is reduced by a Self-organizing Neural Network (SONN), whose neurons correspond to place cells, found in the hippocampus of mammalian brains. Consequently, path planning is done by Dijkstra's algorithm, a complete and optimal graph search that would be exhausting in the full C-space.
- Extensive investigation of six different network architectures for C-space reduction. The generated networks as well as subsequently generated paths were analyzed. The Growing Neural Gas (GNG) showed the best results and was successfully implemented on a real robot.
- Integration of static, and more importantly, dynamic obstacles using Lookup Table (LUT), enabling the fast and efficient transformation from task space to C-space. This allowed the successful implementation of path planning with reactive collision avoiding using a real robot. The proposed system outperforms common sample-based planners like the RRT or PRM showing real-time capability.
- Realization of a benchmark for parallel hardware solutions and respective software tools for simulating SNN. Scientifically interesting is the focus on a robotic use case and the fact that neuromorphic chips, GPU-based solutions and neurostimulators are considered. An approach, developed in the course of this thesis was used. It implements learning by STDP.

The aim of safe and reactive motion planning is thereby not implemented in an all-encompassing system. However, it is addressed on different levels and each of the achievements mentioned above brings science a little step closer to that goal. A special focus of this work is on the development of neural, biologically plausible methods. However, this is complemented by the transfer of these theoretical concepts to real hardware and the analysis of different solutions regarding hardware/software co-design. Thus, this thesis pushes basic research from neuroscience closer to an industrial context with robotic applications.

1.4. Outline

After the introduction, which mainly motivates, classifies and differentiates the thesis from the state-of-the-art research, follows chapter 2, **Foundations**. This chapter is divided into two parts. Firstly, the brain is considered here as a model. Its structure and basic functioning are investigated, followed by a more detailed description of the human visual complex and a consideration of how spatial perception and navigation occur in mammals. Secondly, an introduction to technical concepts is given which apply the principles observed in nature. In particular, the three topics SNN, SONN and event cameras are presented, which all form the basis of this thesis. The main part consists of three chapters, each focusing on one of the research questions from section 1.2. These chapters all embody the same structure, Initially, the state-of-the-art in the respective area is considered, which is concluded with a discussion. Then, an approach to this problem is proposed, which is evaluated in a further section. Each of the three main chapters ends with its conclusion. The chapter 3, **Event-based Stereo Vision**, builds on research question 1. Regarding related work, monocular and stereo techniques are considered separately. However, the focus here is on the latter. The approach presented in this chapter tackles the correspondence problem in stereo vision using event-based data and SOM. A biological motivation, a formalism and some suggestions to extend the basic algorithm are provided. One part of the evaluation takes place in simulation. This includes the qualitative investigation and the proof-of-concept, that the method is suited to solve the correspondence problem. The second part of the experiments was carried out on event-based data from a stereo setup with two Asynchronous Time-Based Image Sensor (ATIS). In chapter 4, **Reactive Neural Path Planning**, the research question 2 is investigated. The state-of-the-art includes both, conventional and brain-inspired methods for path and motion planning. Regarding conventional methods, primarily optimal and sample-based planners are presented. For brain-inspired techniques, a focus is on solutions with SNN. Whereby, there are also two own approaches as part of the related work. Finally, a review of the relevant literature to path planning in the reduced C-space is given. The main part of this chapter embodies three sections. Firstly, a presentation of the basic concept, which includes appropriate training data and learning to represent the effectively used subspace by a SONN. Secondly, a consideration of different network types for SONN is given. These versions have individual advantages and are therefore suited to the task to a varying degree. Thirdly, it depicts how the trained network can be adapted to enable dynamic obstacle avoidance. For evaluation, the different SONN styles are examined first, followed by a comparison of the presented method using a WFA in contrast to Dijkstra's. Then the obstacle avoidance is tested for the most suitable network candidates. The evaluation is completed by a comparative analysis of the presented approach with sample-based planners. In chapter 5, **Neuromorphic Technologies for Neural Algorithms**, research question 3 is looked at in more detail. Different hardware solutions and simulation tools for SNN are presented here first. Followed by a consideration of respective benchmarks in the literature. The core part of this chapter initially introduces the test candidate, an own implementation for a neural WFA using SNN. Subsequently, details regarding the realization of the benchmark and the metrics used are laid out. The evaluation is strongly based on the presented metrics. However, the discussion of this chapter is two-part and includes a context analysis of the presented results as well as a consideration regarding limitations. The thesis ends in chapter 6 **Conclusion**, with a critical assessment of the entire work. The results are recapitulated and contributions are highlighted. In the outlook, planned extensions and possible additions to the work are discussed.

2. Foundations

The enormous progress that has been made in the field of AI has been driven by the attempts and efforts of many scientists to answer the following two questions; "how does the brain work?" and "how can we build intelligent machines?" [63]. Thus, brain theory is the foundation for Artificial Neural Network (ANN). A definition for intelligence is not trivial as it is enormously versatile. Traditionally, board games are seen as a good way to measure intelligence. Less obvious, on the other hand, is motion control of the high-dimensional body. The brain must solve very complex control problems to produce precise, error-tolerant, adaptive movements. It also excels at tasks that do not seem difficult, but are very complex to solve technically, such as listening out one's name in a noisy environment or recognizing emotions only at running patterns. AI has made remarkable progress, especially through deep learning, in some of these fields. For example in the case of alphaGo, the technical solution overtook the brain. However, in many areas, the brain is still almost an unattainable model.
This thesis, while intended for readers with a strong computer science background, deals with a very interdisciplinary field. Therefore, this chapter provides necessary knowledge in biology and computational neuroscience. The division into section 2.1 and section 2.2 is intended to draw a parallel between the model in nature and the technical replica.

2.1. Design and Functioning of the Human Brain

Even though impressive progress has been made in AI over the last decade, in terms of energy efficiency technology is still lagging. The energy consumption of the human brain can be very precisely determined, as its consumption of oxygen and the associated burning of glucose can be measured exactly and converted into an electrical power equivalent [64]. Under provoked limit load the power consumption is estimated roughly between 15 and 20 watts. In comparison with supercomputers, assuming an energy consumption of more than 10 megawatts, this results in a factor of half a million. Hence, our brain requires about 500 000 times less power than such a high-performance computer.

2.1.1. Neurons, Synapses and Plasticity

Most of the current knowledge of brain anatomy and neurophysiology has been developed since about the middle of the 19th century. The cell theory, which is still taught today, was able to establish itself mainly due to the scientific work of Golgi [65] and Santiago Ramón y Cajal [66]. This also forms the basis for the term "neuron", which was coined in 1891. Neurons are the structural elements of the nervous system and thereby the main information processing units. The term "neuron" refers to the nerve cell as well as its processes [67]. According to recent studies, the human brain possesses an average of about 95-100 billion neurons [68]. Neurons are anatomically independent of one another and their processes only interfere with other neurons through impulses transmitted by synapses. This structural isolation has the advantage that if degenerative changes occur in nervous tissue, such as diseases, they are effectively prevented from spreading [67]. In the cortex, there is a multitude of neurons that differ strongly

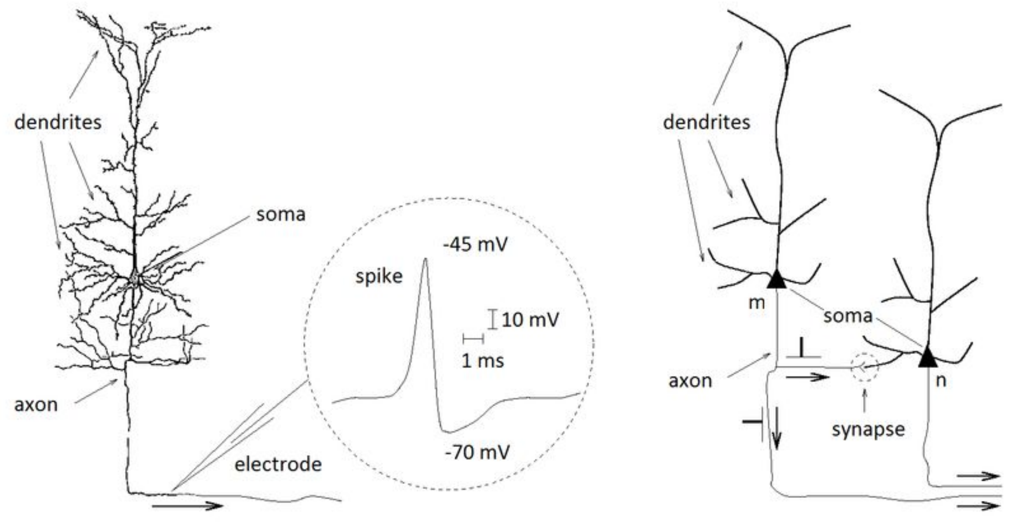


Figure 2.1.: A drawing from Ramón y Cajal depicting the anatomy of a neuron is shown at the left. An exemplary representation of a neural signal, also called action potential or spike, is shown in the center. At the right, a schematic visualization of signal transmission from a presynaptic to a postsynaptic neuron is given. The synapse is marked by a dashed circle. Image source: [69]

in their functionality, but the basic structure is uniform. As shown in Figure 2.1, a neuron consists of three components, the cell body called soma, the dendrites and the axon [69]. The processing of incoming stimuli takes place in the soma. While the dendrites receive incoming signals and transmit them to the soma, the axon is used to transmit signals leaving the neuron. The synapse is the connection to other cells, thus the junction between two neurons [70]. The soma only generates an output signal if the membrane potential exceeds a certain threshold.

In each signal transmission between two neurons, there is always a presynaptic cell sending the signal, and a postsynaptic one receiving it. However, this is not a 1-to-1 relationship, on the contrary, a single neuron of the vertebrate cortex can address up to 10^4 postsynaptic neurons. The majority of the receiving cells are located close to the sending neuron, but in some cases, the axon can also reach more distant brain regions, or even the entire body [71; 69]. In contrast, dendrites, even though they have a complex and highly branched architecture, affect only the immediate vicinity of the neuron [72]. The neuron is separated from its surroundings by the cell membrane. Due to specific channels, the membrane is permeable to special types of ions, electrically charged particles, most prominent sodium (Na^+), potassium (K^+), calcium (Ca_2^+), and chloride (Cl^-). Information processing by neurons occurs mainly through electrical and chemical signals [71]. To produce electrical signals, neurons change their membrane permeability to certain ions. The delta of certain ion concentrations between the neuron and its environment creates a polarization, called the membrane potential. Whenever a neuron is not signaling, its membrane potential is at its resting state. Thus, its potential difference to the cell's surroundings is between -30 mV to -90 mV, which is referred to as polarized. The membrane potential can now change in two directions, due to signals from other neurons. When it becomes more positive it is called depolarization and when it becomes more negative it is called hyperpolarization [73]. The neural membrane potential is under the influence of the ion channels, which, by their opening, allow certain ions to enter the cell. However, as these channels are ion-specific and also voltage-gated, the membrane potential, along with both internal and external signals, affects their opening [71]. For sodium-specific channels, that means that if the membrane potential surpasses a threshold they will open, allowing even more sodium to enter the cell, which causes the membrane potential to increase further. In contrast, potassium-specific channels, which open at a very high membrane potential lead to an outflow of potassium ions and consequently to a strong decrease of the membrane potential. Hence, sodium-specific channels reinforce an existing depolarization through a positive feedback process while potassium-specific channels reverse a strong depolarization into a hyperpolarization [71; 73]. Both mechanisms together lead to short electrical pulses called action potentials or spikes, as shown in the dashed circle in the center of Figure 2.1. Spikes are the basic communication mechanism of neurons. Their duration of 1-2 ms and amplitude of ca. 100 mV remain consistent when propagating along the axon to other neurons [69]. Immediately after the neuron has initiated a spike, it is unable to do so again. These few milliseconds are called the absolute refractory period, followed by the relative refractory period, which can last up to 10 ms and makes it difficult, but not impossible, to generate action potential [71]. A series of spikes, emitted by one single neuron in a short period, is called a spike train. As action potentials are in principle binary, a spike is either generated or not, but it does not vary in its duration or amplitude, the informational value lies in the number and the exact time of occurrence. Due to the refractory period, spikes are prevented from overlapping or merging into each other [69].
A synapse is a point where the axon of a presynaptic neuron interfaces with sev-

eral dendrites of a postsynaptic neuron [69]. As the brain is a highly connected structure, each neuron has 5,000 – 10,000 synapses which transmit spikes between neurons and thereby affect their membrane potential [74]. Usually, synapses form the middle structure between a presynaptic axon and a postsynaptic dendrite [72], however, it can also occur that an axon is connected directly to a soma or another axon by a synapse [73]. The brain has approximately 100 trillion synapses which only forward nervous impulses in one direction [67]. Even

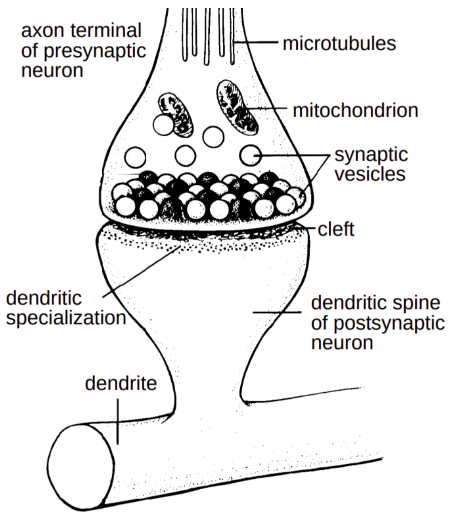


Figure 2.2.: Schematic drawing of a biological synapse. At the top, the axon terminal of the presynaptic neuron, holding several synaptic vesicles filled with neurotransmitters, is shown. At the bottom is the dendritic spine of the postsynaptic neuron which is separated from the axon terminal by the synaptic cleft. If a spike from the presynaptic neuron is propagated through the axon, the neurotransmitter is released and surpasses the synaptic cleft to bind to receptors of the dendritic spine. Image source: [71]

though electrical synapses, directly transmitting action potential, exist [73], by far the most common type in the human brain is the chemical synapse, visualized in Figure 2.2. Thereby the axon terminal is only separated from the signal-receiving or presynaptic neuron by a small gap, the synaptic cleft. When a spike from the axon reaches the synapse a chain reaction of biochemical processes is initiated resulting in the release of neurotransmitters into the synaptic cleft. Specialized receptors of the postsynaptic cell membrane recognize these transmitter molecules and initiate the opening of ion channels, thereby allowing ions from the surrounding to enter the postsynaptic neuron. This causes a change in the membrane potential of the postsynaptic cell triggering an electric reaction, which is called the Postsynaptic Potential (PSP) [71; 69]. Depending on the neurotransmitter released, the PSP differs. If the transmitter opens sodium channels, the inflowing ion increases the membrane potential (depolarization), called excitatory PSP. However, if potassium-specific channels are opened by the transmitter,

the membrane potential is lowered by the potassium leaving the cell (hyperpolarization), called inhibitory PSP [73].
The amplitude of a postsynaptic response to a presynaptic action potential is determined by two factors. On the one hand, the number of neurotransmitters of the synaptic vesicles that are released into the synaptic cleft. On the other hand, the number of receptors that the postsynaptic neuron holds. Both parameters, collectively referred to as synaptic strength or synaptic efficacy, can change over time and depending on external influences. A modification of the synaptic strength is called synaptic plasticity [75; 76]. Hence, synaptic plasticity modulates how neurons communicate via synaptic transmission by changing the transmission efficacy of existing synapses. Concerning its temporal effect, synaptic plasticity is differentiated. First, short-term plasticity with a usual duration of action ranging from milliseconds to, in exceptional cases, a few minutes [77]. It directly affects neural computations and essentially relates to stimulus-driven activity [78]. Second, long-term plasticity, lasting for 10 minutes or more, is the neural process underlying learning and memory. It also plays an important role in the development and structure of the nervous system of adolescents [79]. There are two

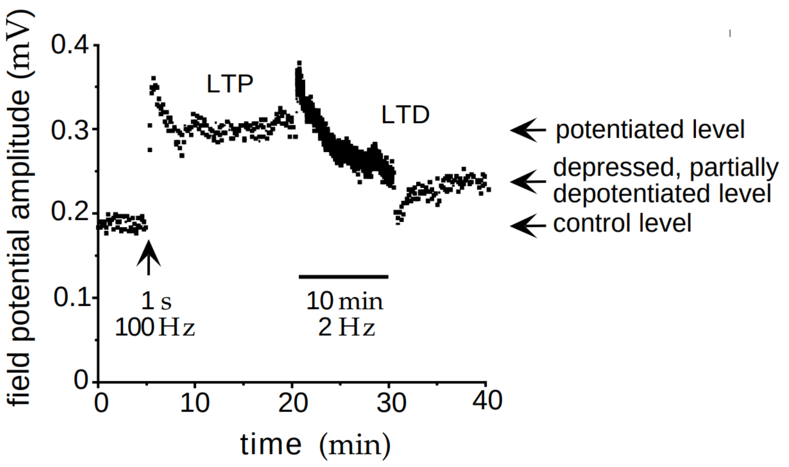


Figure 2.3.: Hippocampal occurrences of LTP and LTD in a rat. After five minutes (see arrow), a 100 Hz stimulation applied for one second induces a big change in the potentiation level. As it remains for about 15 minutes, it is considered LTP. Subsequently, 2 Hz is applied for about 10 minutes, causing an amplitude reduction of the response. The level of reduced potentiation is referred to as LTD. Image source: [71]

kinds of short-term synaptic plasticity. First, synaptic depression is the progressively stronger decline of a postsynaptic response to ongoing presynaptic activation. And second, synaptic facilitation increases the postsynaptic response with repetition [78]. However, both mechanisms do not cancel each other out but rather are modeled in a superimposed manner [80]. Short-term synaptic plasticity functions in the mammalian brain as a kind of filter, as it alters temporarily the way synapses process information [76]. Synapses that only receive low-frequency input act as high-pass filters. Therefore, high-frequency spike trains are transmitted with higher efficacy. Respectively, synapses that originally received at high-frequency function as low pass filters, transmitting low frequent activity unaffected, while inhibiting high-frequency spike trains [81]. The majority of processes regarding short-term synaptic plasticity have a fundamentally similar sequence of events. They are initially triggered by short frequent spike trains that increase the calcium level in the presynaptic axon terminal. This raises the probability of neurotransmitter release [76]. Both depression and facilitation appear in two forms, paired-pulse plasticity, particularly short-lived, and trains of stimuli, a slightly more long-lived form [77]. Regarding long-lasting plasticity, a distinction is also made as to whether the effect on the synapses is excitatory or inhibitory. Long Term Potentiation (LTP) increases synaptic strength over time while Long Term Depression (LTD) decreases it. For an illustrative example of the biological occurrence of LTP, and LTD see Figure 2.3. Both forms of plasticity can be extremely persistent and may even last for several years. This differs greatly in individual cases, but in general, as long as experience and training are retrievable [71]. Already in 1949, Donald Hebb found that synaptic strength changes according to the temporal relationship of the fire behavior of different cells. He stated that *"when one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell."* [82]. Thus, Hebb found that synapses become more effective

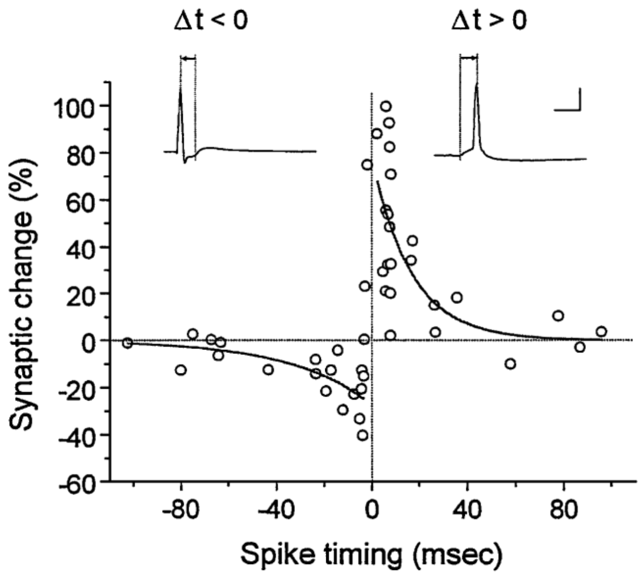


Figure 2.4.: Asymmetric temporal window for effective spike occurrences. Excitatory postsynaptic current is plotted against precise spike timing. The graph shows a curve representing LTD on the left and LTP on the right, generated by the temporal correlation of pre-and postsynaptic spikes of synapses in the hippocampus. At the top, the respective windows for LTP and LTD are visualized. Image source: [83]

when the presynaptic neuron spikes often just before the postsynaptic one. In literature the popular phrase *"fire together wire together"* is often used in this context. However, as Hebb was a psychologist his concept of learning is rather vague and his transcripts lack precise approaches and formulas. In [84] Hebbian learning is defined by two underlying assumptions:

1. Learning is strongly modulated by local information of the pre- and postsynaptic neurons.
2. Learning depends on the correlation between the processes of these neurons, which is reflected in their weights.

Even though Hebb's original postulate only includes LTP, learning can only be achieved through a combination of strengthening and weakening synapses, to prevent saturation [85; 83]. The exact nature of the mechanisms involved in synaptic plasticity is not yet fully understood, and still an active field of research. A relatively new observation, which emerged in the late 90s, is, that for synaptic plasticity the precise timing of spikes is very important [86; 8; 7; 87]. This is further emphasized by the discovery that stringent windows, in which the occurrence of a spike induces a certain reaction, exist [88]. As shown in Figure 2.4, these temporal windows are usually, but not always, asymmetric [83]. However, in [89; 90] experiments are presented suggesting that the precise spike-timing and its effect on synaptic plasticity are part of a bigger picture.

2.1.2. Self-Organization in the Brain

Neurobiological learning paradigms apply self-organized processes. These are based on observations by scientists that in some cognitive processes learning is controlled unsupervised by experience and external input [91]. Respective brain structures, referred to as Self-Organizing Systems (SOSs) [92], are already present before birth [93; 94], but detailed self-organized areas are educated throughout life [95]. Thereby, feature maps of the mammalian cortices which are established as synapses build up the neighborhood- or topology-preserving maps [96]. Within the sensory cortex, this form of topographic organization of adjacent nerve cells is often triggered by neighboring cells within the sensory input space [97]. An SOS is a self-containing, internally regulated structure that emerged through evolutionary processes such as mutation and selection. A very interesting property of an SOS is the ability to react to external influences and still restabilize itself [92]. The underlying architectures of SOSs are referred to as brain maps in literature [98; 99], describing the brain's self-organization regarding spatial representations. Hence, mappings in the sense of topologically ordered representations of features [96]. Besides Penfield's and Rasmussen's well-known Homunculus [99] examples include the tonotopic map discovered in cats' auditory cortex [100], the gustotopic map located in the primary taste cortex [101], the whisker map found in rodents' barrel cortex [102] and a map of the same name in the somatosensory cortex of primates [103]. By artificially visualizing the blood flow, oxygen uptake, electric current or phosphate metabolism, for example, using gamma cameras or magneto-encephalography, brain maps can be observed [98]. Malsburg [104] and Willshaw [94], used the ordered projection of the retina to the visual cortex as a template. In their prototypical design, the retina and the cortex are each represented as a sheet of cells. Thereby, neighboring presynaptic cells within the sheet representing the retina project on neighboring postsynaptic cells of the sheet representing the cortex. Connections of the postsynaptic sheet are excitatory between closely neighbored cells to strengthen the activity additionally. Respectively, inhibitory connections join cells farther apart, to prevent neural activity. These two mechanisms generate continuous mapping.

2.1.3. Human Vision

The Retina

The human retina, the brain's source of visual information, is a multi-layered neuronal network located in the back of the eye. The retina is not only responsible for data acquisition but also for encoding and transmitting information to the brain [105]. It achieves that by converting spatiotemporal illumination information into pulses [106]. Of the many layers of the retina, the photoreceptor, the outer plexiform and the inner plexiform layer are mainly responsible for the processing [105; 107]. As visualized in Figure 2.5, each layer contains special cell types, crucial for the function of the layer. Photoreceptors, the light-sensitive cell types, build up the photoreceptor layer while bipolar cells are located in the outer plexiform layer and ganglion cells in the inner plexiform layer. Photoreceptors, which absorb light and convert it into an electrical signal that triggers the release of neurotransmitters [106], can be divided regarding which wavelengths of light they react. Cones are responsible for color vision and specialized sub-types for the different frequencies of red, green and blue light exist [108]. The other photoreceptor of the retina, the rod, is responsible for motion detection, night vision and peripheral vision. This cell type is much more prevalent in the retina with 120 million rods as opposed to 6 million cones [105]. Both types of photoreceptors are at their resting potential, the non-excited normal state, if no light penetrates the eye. These cell types are connected to bipolar and horizontal cells which are further connected to amacrine cells and ganglion cells. While at their resting potential photoreceptors release neurotransmitters exciting bipolar cells in the outer plexiform layer, which subsequently inhibit ganglion cells in the outer plexiform layer also through neurotransmitters. Hence, in darkness, photoreceptors and bipolar cells are active while ganglion cells are inactive. If light enters the eye, however, the process just described causes ganglion cells to be excited and, by increasing their action potential, stimulate the visual center of the brain via the optic nerve [109]. The components visualized in Figure 2.5 build up a pipeline for visual preprocessing, transforming simple dots to more complex visual elements embodying shapes and motions [110]. In this context, photoreceptors can be seen

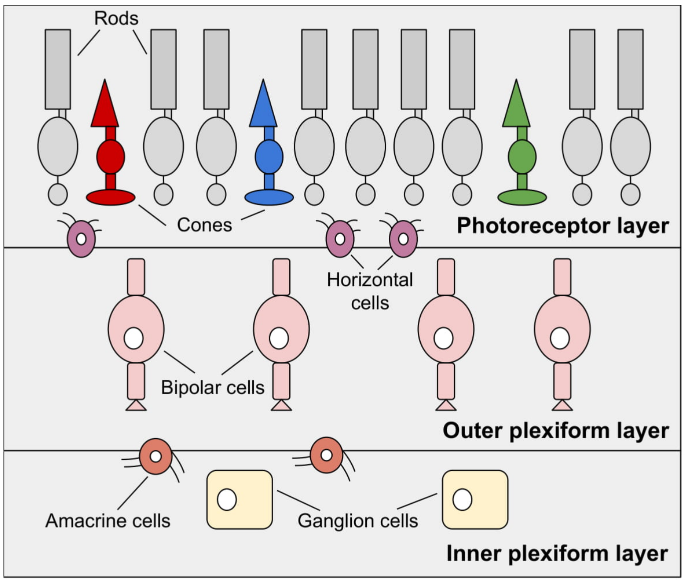


Figure 2.5.: Simplified sketch of the human retina reduced to three layers, the photoreceptor layer, the outer plexiform layer and the inner plexiform layer. The outer plexiform layer embodies bipolar cells and the inner plexiform layer embodies ganglion cells. Supplemented with the layer-connecting horizontal and amacrine cells. Image source: (Steffen et al. 2019c)

as pixels generating input for neural pathways. Particularly important for human vision is the parvocellular and the magnocellular pathway. As neurons of the parvocellular pathway are sophisticated regarding high spatial resolution but have a relatively low temporal resolution, they are responsible for details and color perception. Neurons within the magnocellular pathway are oppositely specialized, with low spatial resolution and a high temporal resolution, which make them well-suited for detecting motion and depths. Although such specializations exist, there is not, as originally assumed, a clear division of responsibility. Instead, an integrated projection from all pathways into the visual cortex is responsible for all visual sub-tasks [111; 110; 105].
As well bipolar cells as ganglion cells can be divided into two classes, the neurons that code for a positive light difference, the ON-types, and those that code for a negative one, the OFF-types. If the illumination is uniform over some time, the membrane potential of both neuron types is at their resting state, however, an increase in lightening stimulates ON-types while a decrease stimulates OFF-types [108]. This effect is handled by horizontal cells which connect photoreceptors and bipolar cells laterally. Thereby, horizontal cells compare each new signal from the photoreceptors to a representative value that represents a mean. Amacrine cells are inhibitory interneurons that transmit signals between bipolar and ganglion cells [106]. The retina implements multiple design principles [107; 112; 113; 106] to increase the quality and flexibility of signal processing while guaranteeing efficient coding with as little information loss as possible. The fol-

lowing list is taken from (Steffen et al. 2019c):

Design Principal 1. *Local automatic gain control*
The accuracy is flexibly adapted to the scene instead of measuring changes in lightning in terms of absolute values. This is realized through time-space bandpass filtering and adaptive sampling at the photoreceptor level. As a consequence, a larger dynamic range is achieved, thus, the ratio between maximum processable signal and background noise increases.

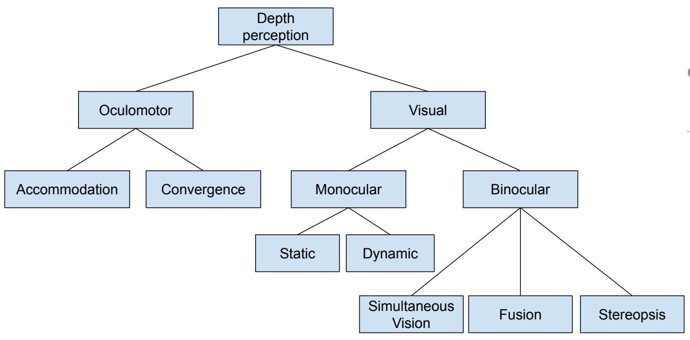
Design Principal 2. *Bandpass spatiotemporal filtering*
Suppressing outliers in the frequency in the outer plexiform layer has two beneficial effects. Redundancies caused by low frequencies are discarded and motion blur caused by high frequencies is suppressed. This is further enhanced by high-pass filters located in the inner plexiform layer.

Design Principal 3. *The equalization of the signal*
Distinguishing bipolar and ganglion cells in ON- and OFF-types lowers the spike rate and thus, decreases the data throughput. This represents a very sparse coding form.

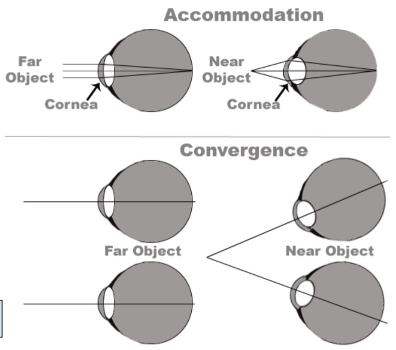
Design Principal 4. *High spatial and temporal resolution*
The retinal pathways of sustainable neurons, the parvo cells, and volatile neurons, the magno cells, induce a spatially high and temporally low resolution in the center of the visual field. The reverse effect is true for its corners.

Depth Perception

The human eye projects the 3D environment onto a 2D image and the exact position of objects in space is lost. Estimating distances, which is essential for various manipulation tasks and safe navigation in unknown environments, is only possible for us as humans because we can reconstruct depth information from 2D images, and are thus capable of 3D perception [109; 114]. There are many mechanisms involved in human depth perception which can be roughly divided into oculomotor and visual depth stimuli [115; 116], as shown in Figure 2.6(a). For oculomotor depth stimuli, which are useful for viewing at close distances, the position of the eyes and the tension of eye muscles is crucial. For close objects, the muscles are tense, while for more distant objects they are relaxed. Thereby, two techniques are distinguished, convergence for distances up to 600 cm, and accommodation used for ca. 20-300 cm, as shown in Figure 2.6(b). Convergence



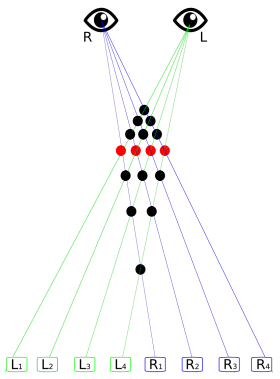
(a)



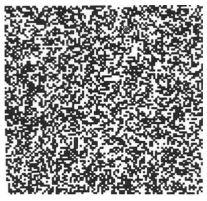
(b)

Figure 2.6.: (a) An overview of the different mechanisms involved in human depth perception, which can be divided roughly into oculomotor and visual depth criteria. (b) Oculomotor or physiological stimuli are obtained by accommodation and convergence. Image source for (b) is [117].

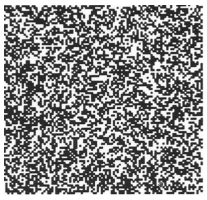
consists of the movement of the eyes toward the center when viewing nearby objects, however, only two-thirds of the population use it as a cue to depth [118]. Neural mechanisms for depth perception vary for individual humans. Accommodation results from the change in the shape of the crystalline lens when objects at different distances are focused [119]. Visual stimuli, which can be divided into monocular and binocular, make a far greater contribution than oculomotor depth stimuli in human 3D reconstruction [109]. Monocular vision refers to all the information we can extract from a simple 2D image to understand a scene. First, static cues like the knowledge about the common shape and size of an object as well as texture and shadows. Also, occlusions can be used to determine the arrangement of objects. In addition to static monocular vision, there is dynamic monocular vision. It arises from motion-induced depth stimuli generated by head and eye movements. Besides objects being covered and uncovered due to the observer's motion, this also includes parallax, which occurs if an observer moves in parallel to several objects that are located at different distances. Thereby the observer perceives objects to move slower if they are located further away [120; 109; 121]. Binocular vision is the ability to obtain the 3D shape of an object through the interaction of both eyes. It is caused by the differences between the images of both eyes of the same scene and comprises simultaneous vision, fusion and stereopsis. As the sensitivity is higher [122] and latencies are shorter [123] for binocular vision, compared to monocular, this method is more robust and precise. Simultaneous vision, inhibiting false visual sensations and disturbing artifacts as well as fusion, necessary to avoid double vision [120; 109], only plays a minor role. In contrast, stereopsis also referred to as stereoscopic depth perception, is especially powerful because two perspectives allow a more precise computation of the distance to objects [126]. The correspondence problem, shown graphically in Figure 2.7(a), poses the issue of which points of the left and right eye refer to the same point in the real world. A horizontal displacement



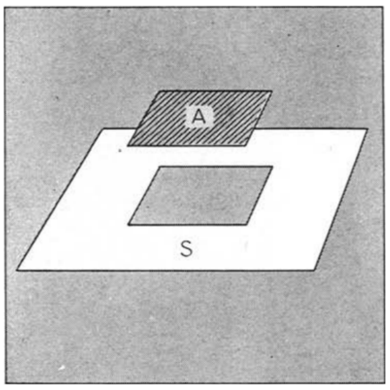
(a)



(b) left



(c) right



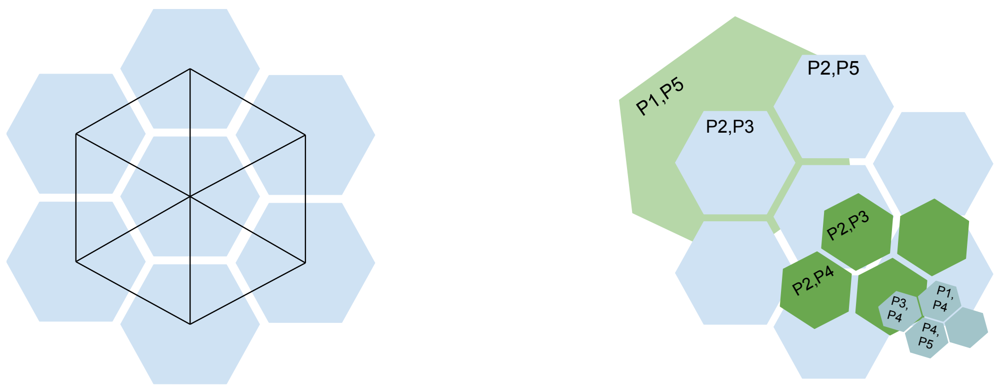
(d)

Figure 2.7.: (a) Ambiguities in retinal projection cause the correspondence problem. Four objects are captured by the eyes from two slightly different perspectives, shifted laterally. $L_1 - L_4$ represents how the objects are perceived by the left eye and $R_1 - R_4$ by the right eye respectively. The dots on the rays mark all possible matches, but only the red dots are correct. (b) & (c) almost identical images with random dots. (b) is created from (c) by cutting out a square of dots and shifting it to the side, the size of the shift determines the perceived depth. The empty space is refilled with random dots. (d) Participants perceived the shifted square as if it was higher above the rest of the image. (a) is adapted from [124]. Image source for (b) & (c) is [125] & for (d) is [117].

is created by the ca. 6 cm shift regarding the view angle of both eyes. The resulting difference in image location perceived by the left and right eye is known as disparity. Since disparity, is inversely proportional to the depth it can be used to reconstruct 3D objects from two 2D images, thus solving the correspondence problem. Thereby, the corresponding dots of both images must be determined to obtain disparity [120; 109; 125]. Even though binocular vision is a very old research topic, the association between spatial depth and disparity is relatively new. As stated in [127], two developments in research regarding binocular vision have contributed significantly to that. First, the invention of the stereoscope by Wheatstone [128], opened up new possibilities for experimental research. Second, Julesz's separation of depth perception and object recognition by the use of a Random-dot Stereogram (RDS). Humans solve the correspondence problem, shown in Figure 2.7(a), reliably. However, Julesz was able to prove that the perception of spatial depth takes place in the brain and not in the eyes [125; 114]. Before, the common scientific understanding was that the images of both retinas were processed separately by the visual system. Hence, the theory was that depth reconstruction only follows scene segmentation and object recognition which was allegedly executed individually for each eye. The breakthrough revelation Julesz made was that stereopsis works perfectly well with an RDS, which proved that it does not use object recognition as a basis. In [125; 114] it is presented, how the human brain solves the correspondence problem. In experiments, Julesz showed RDSs, identical images of dots with a shifted square, to his participants, as shown in Figure 2.7(b) and (c). The test participants were able to recognize a depth map as shown in Figure 2.7(d). This effect is caused as the brain tries to reconcile the two images, but there is a height difference. That humans can see depth in an RDS shows also that binocular disparity is an especially strong depth stimulus, as it does not require any monocular cue. Research, based on the invention of RDSs, led to one of the most influential books in cognitive science and the foundational work for stereo vision [129]. However, the exact mechanisms of how the brain establishes connections between two dots, thus solving the correspondence problem, is still an active field of research.

2.1.4. Spatial Awareness in the Brain

For achieving a deeper understanding of the sense of orientation, present in humans and animals, researchers have conducted a great variety of experiments and presented several theories [41]. Behavioral research on spatial navigation has shown that besides navigating great distances, mammals and also smaller species, navigate flexibly and very efficiently in cluttered surroundings. An internal spatial representation of the environment is somehow present in the brain. Edward Tolman supported this observation with experimental evidence on spatial navigation in rats and introduced the name "cognitive map" in this context [40]. A widespread brain network is involved in spatial representation, direction and orientation, but, spatially modulated neurons, located in the hippocampus, play a major role [42; 41]. A variety of different cells interact here, however, place cells, head direction cells and grid cells have been identified as particularly important and therefore the cognitive map's neural basis [41]. Even though many other cell types have been discovered, their functionality is not as well understood and in some cases intertwined with other parts of the brain. However, many details regarding the form and structure of this hippocampal spatial map, such as distance metrics, landmark construction and the applied coordinate system, are still under discussion. Furthermore, some voices in the scientific community still doubt the assumption of a map because, according to their findings, place cells are arranged as a memory. Hence, instead of marking a specific spatial point within a map like a brain structure, previous events are recalled [131]. A place cell is a specialized cell whose activation depends on spatial location. It encodes for specific spatial areas referred to as place fields, thus its spiking intensifies in case the animal enters its respective place field in the real world [132]. A Gaussian function can be applied to approximate place fields [133], whose size can be in the range of 20 cm to several meters, depending on where their representation is located in the brain [130]. Place cells do not represent space topographically, thus, neighboring cells can encode for distant regions of the environment and a location may be represented by a group of non-adjacent neurons [134]. Even though place cells have first been discovered in rats [132; 42], similar cell types



(a) Grid pattern (b) Interaction of grid and place cells

Figure 2.8.: (a) The most common pattern of grid cells is the hexagonal blue structure. However, as the space in between these cells can be approximated by a triangle, this is also used as a description in literature. (b) Simplified relationship between grid and place cells as explained in [130]. The place cells P1-P5 are related to grid cells which belong to several grids varying in resolution and orientation. Adapted from: (Augenstein 2021)

were also found in other animals [135]. As place cells by themself do not explain a spatial cognitive map, their existence in the hippocampus was doubted by many researchers [136]. Thus, the interaction of place cells with other neuron types was studied extensively and led 2004 to the discovery of grid cells as they showed comparable spike patterns [137]. While place cells are located in the hippocampus, grid cells are part of the medial entorhinal cortex [138; 130]. They are arranged in a regular, grid-like pattern mosaic representing any 2D surrounding the animal has previously entered. In literature, two ways to describe the pattern are used a periodic triangular grid [138], or a hexagonal pattern depending on the perspective, as visualized in Figure 2.8(a). Grid cells are generally active in all environments, the information concerning the spatial representation is obtained by which ones are activated simultaneously [138]. In contrast to visual sensory processing, place cells do not spike for one specific input cue, instead, they can only be triggered if several different cues are associated [42]. Based on neurophysiological research and anatomical connectivity it is assumed that grid cells are the main input source of place cells [130]. Thereby, 10-50 grid cells' summed activity builds up one single-place field. However, each grid cell might be the input source of several place fields. Grid cells originate from several overlaid grids with diverse orientations, resolutions and spatial displacement, as shown in Figure 2.8(a). The movement of the animal is thus followed in several grids at the same time [138]. A symbolic representation of this theory of how place and grid cells are connected is visualized in Figure 2.8(b). Preliminary results imply many constraints on how grid and place cells are connected, however, respective research is still very much at the beginning [130]. A scientific consensus is reached that place cells are not autonomous, but combine signals of several different grid cells from grids with different specifications. Hence, an animal's position is encoded in multiple layered grids and the connection to place cells decodes this relationship. Finally, head direction cells work similarly to a compass and provide information about an animal's orientation without additional sensory input [139].

2.2. Modeling the Human Brain

No matter how powerful an algorithm is, it must be implemented on hardware and many applications like mobile robots, cars, phones and drones have limited energy resources. In addition, the ever-increasing global energy consumption and its consequences for the climate also require more resource-friendly solutions. The question remains, how to implement algorithms in an energy-efficient way for versatile applications. While algorithms often mimic the brain, up until recently, hardware was developed without considering the design of nature. Today's hardware is mostly sequential, in contrast to the highly parallel brain which solves dynamic processing through action potentials.

2.2.1. Networks of Spiking Neurons

Spiking Neural Network (SNN) are biologically more accurate as they supplement ANN with a temporal component [4; 5; 6]. This brings new challenges and thus new solutions to neural learning as well for software and hardware. The technical development of SNN can be roughly divided into three phases, as visualized in Figure 2.9. The logical building blocks, referred to as the neuron model, and the mode of information transmission as well as the network structure differ significantly for each. The 1st generation is based on McCulloch-Pitts neurons

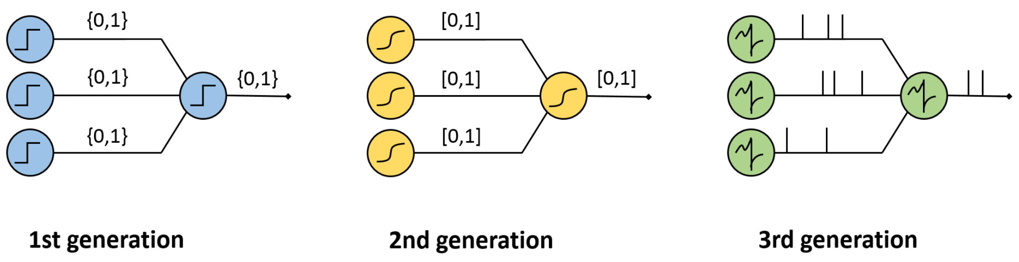
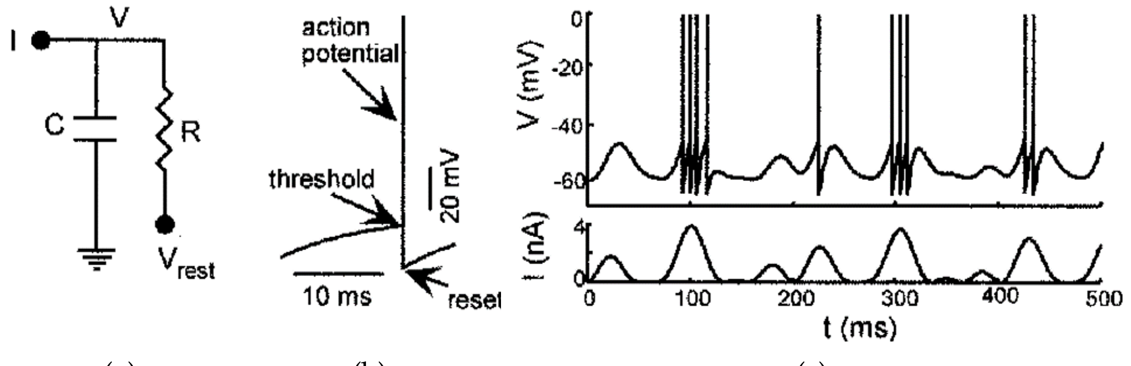


Figure 2.9.: The computational units and the mode of information transmission for the three generations of ANN. The 1st generation applies McCulloch-Pitts neurons in a multi-layer perceptron only transmitting binary values. While neurons of the 2nd generation use a continuous activation function and can handle numerical data. The 3rd generation processes temporal information of individual spikes, making it most similar to the biological model.

which are also referred to as perceptrons or threshold gates [4]. They are organized in a multi-layer perceptron only transmitting binary values. Neurons of the 2nd generation use a continuous activation function and can handle numerical data. The 3rd generation processes temporal information of individual spikes, making it most similar to the biological model. SNN are specially designed to deal with time-related data, thus, spiking neurons unlike analogs, consider the temporal dynamics. While neurons of a traditional ANN are based on a differentiable activation function, in spiking neurons the membrane potential evolves in time depending on the input of weighted spikes. This causes complex network structures instead of clean layers, leading to very powerful systems but requiring more complex learning algorithms.

Neuron Models

Neuron models are an abstraction of the biophysical properties and processes of neurons introduced in subsection 2.1.1, strictly speaking, an equivalent circuit for the selective ion channels. In the electrical circuit, all channels of one ion type are combined into one conductance, referred to as *leak*. The conductivity is recorded as an invariant ohmic resistance R and the electromotive force E is modeled by a battery in series [64]. However, the different neuron models differ greatly from each other, both in terms of biological plausibility and performance. The most commonly applied neuron model is the Integrate-and-Fire (IF). Surprisingly, this model was developed in 1907 [140], long before neuronal mechanisms responsible for the action potential were explored. Hence, Lapicque modeled this phenomenon without any deeper knowledge about the biophysical basis [141]. As a result, the shape of the spikes or even processes like the refractory period are not represented in the model [142]. This neuron model is characterized by a simple electrical circuit and a focus on precise spike time. The electric circuit, shown in



(a) (b) (c)

Figure 2.10.: (a) electric circuit of an IF model consisting of a parallel capacitor and a resistor. (b) An action potential generated by charging the capacitor over a threshold. (c) The system's reaction to a time-varying input current. Image source: [141]

A in Figure 2.10, only embodies a capacitor and a resistor in parallel. They are to artificially reproduce the capacitance and leakage resistance of the cell membrane respectively. A simple circuit like this is not able to reproduce action potentials, however, Lapicque argued that when the capacitor is charged above a threshold an action potential is generated and the capacitor is discharged as shown in B in Figure 2.10. The IF has received numerous extensions and variations over the years [143] for example concerning the role of inhibition or PSP. The simplest form, is the Leaky-Integrate-and-Fire (LIF), formalized as [143; 69]:

$$\begin{aligned} \tau_m \frac{du}{dt} &= -I_{leak}(t) + I_{syn}(t), \\ I_{leak}(t) &= u(t) - u_{rest}, \\ I_{syn}(t) &= RI(t), \end{aligned} \tag{2.1}$$

whereby, the neuron's membrane time constant is τ_m, the membrane potential u, the resting potential at equilibrium u_{rest}. Thus, the LIF is given by I_{syn}, a current representing incoming spikes, from which the leakage I_{leak} is deducted. Hence, input currents are integrated by the membrane potential and part of it is leaked by the leak conductance and finally an idealized action potential is emitted by a threshold mechanism [69]. Even though it ignores the morphology of neurons completely [143] is widely used [69]. A powerful but simple and general model is the Spike Response Model (SRM) introduced in [144]. It is a generalization of the LIF model, giving a simple description of the generation of action potentials while reproducing neuron activity on a purely mathematical basis. For the LIF model voltage is formulated through differential equations. In contrast, the SRM is formalized by the use of filters. The biggest difference in practical use, however, is that the SRM maps the biological refractory period. An extremely realistic model that requires great computing times is the Hodgkin-Huxley (HH) model, presented in [145]. It is an artificially quite accurate replica of the giant axon of a squid neuron. It includes complex physiological features like the three ion channels discovered in experiments. Thus, it generated action potentials with shape, amplitude and duration similar to the ones emitted by biological neurons. The HH model is also based on an electrical circuit, however, this depicts the membrane potential in a more detailed manner and takes into account the dynamic nature of the different ion channels found in the soma and dendrites. The current $I(t)$ embodies the capacitive current I_C and additional components I_K which are summed up over all ion channels formalized as [69]

$$I(t) = I_C(t) + \sum_k I_K(t). \tag{2.2}$$

In the original version, this includes three types, sodium (Na^+), potassium (K^+) and a leakage channel L. Thereby, the currents regarding the three types are formulated as [145]:

$$\sum_k I_k = g_{\text{Na}^+} m^3 h (u - E_{\text{Na}^+}) + g_{\text{K}^+} n^4 (u - E_{\text{K}^+}) + g_L (u - E_L) \tag{2.3}$$

As the availability of computational resources has improved greatly in recent years, efficient simulation of such a complex computational model is no longer impossible. Nevertheless, this model is rarely used. Reasons include its difficult handling as it is not easy to develop an intuitive understanding of its neuronal dynamics. Furthermore, the enormously large parameter space and the fact that such models cannot be analytically explored pose major problems in practice [143]. The Izhikevitch [146] neuron model is a compromise between biological plausibility and computational efficiency. It is, like the IF, a phenomenological model, meaning it only depicts a neuron's input in terms of currents and its output in terms of spikes. Any electrochemical reactions or characteristics of the action potential, as described in subsection 2.1.1, are hereby neglected. However, the Izhikevitch model is a reduced derivation of the HH-model, but it should be noted that it is closer to the LIF in terms of biological realism. Its simulation is very performant and it can reproduce some naturally occurring spike behavior, but not as comprehensively as the HH model. For modeling it applies a differential equation in 2-dimensions. Researchers concluded that the characteristics of an action potential, like its shape, are irrelevant. This is motivated by the observation that spikes do not differ and the information lies solely in their precise timing. Therefore, it is not surprising that many scientists get by with simple phenomenological neuron models since the exact spike, times which hold all relevant information, is also well implemented here.

Learning with Spiking Neurons

Learning in ANN and a sense also in SNN is just adjusting connection weights in a large graph [147]. Despite great efforts by scientists to find performant and biologically plausible algorithms, there are only a few candidates that meet these requirements [84]. The two methods that have received the most attention, are the back-propagation [148; 149] and Hebbian learning [82]. Interestingly, they have an opposite principle of operation as learning in Hebbian learning takes place locally and in backpropagation globally, through the back transport of the error. Also, Hebbian learning is biologically plausible, as introduced in subsection 2.1.1 including a graphical representation of its biological occurrence in Figure 2.4. With the local properties, post-synaptic activity y_i, presynaptic activity y_j and synaptic strength between these neurons w_{ij} it can be formalized as:

$$\Delta w_{ij} \propto y_i y_j \tag{2.4}$$

Regarding supervised learning, no method has as many successful applications in the most diverse areas as the stochastic gradient descent back-propagation [84]. It is formalized as

$$\Delta w_{ij} \propto B_i y_j, \tag{2.5}$$

whereby, B_i is the error that is back-propagated through the network to the neuron i [149]. In stark contrast to Hebbian learning, back-propagation is not biologically plausible. Hence, at least its original version is not applicable for

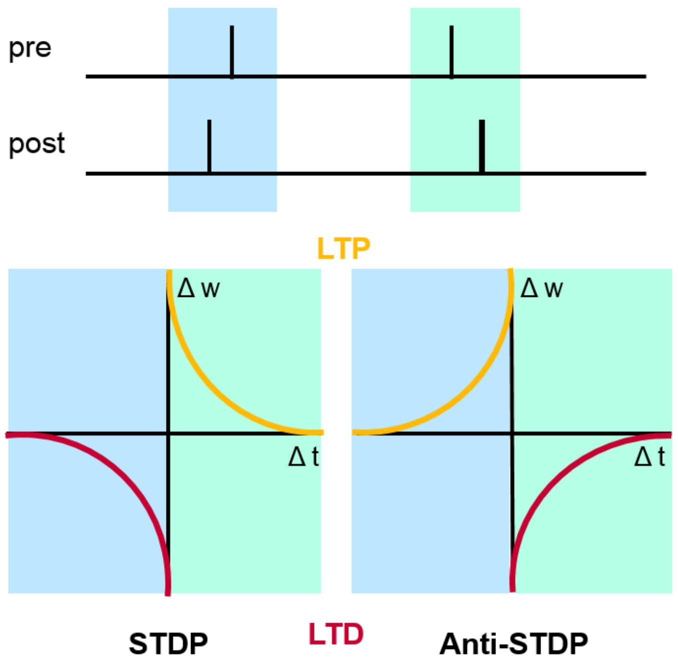


Figure 2.11.: The learning rule STDP and Anti-STDP strengthen synapses according to precise spike timing. Exactly opposite events lead to the same result in both rules.

SNN [150]. That back-propagation originally could not be applied to SNN is caused by the non-linearity of spike-based communication making the neurons non-differentiable. Moreover, this is because the continuous updates, that are part of standard SNN dynamics interfere with alternating forward and backward phases. Also, the history of neuronal activity cannot be stored in the neuron due to memory constraints [151]. However, in a step-wise evolution, several of the issues, regarding the back-propagation algorithm for SNN discussed in [151] have been gradually solved. This development moves from the SpikeProp [152] via the eRBP [153] and the SuperSpike [154] to DECOLLE (Kaiser, Mostafa, and Neftci 2020).
Even though it is inferior to back-propagation in terms of performance, Hebbian learning forms the basis of many learning theories. While back-propagation is a precisely defined algorithm, Hebbian learning is a somewhat soft conceptual term, for example, no equations are used in Hebb's original work [84]. As LTD and LTP build the basis for learning and memory and short-term plasticity only deals with sensor-related reactions, as presented in subsection 2.1.1, long-term plasticity is the more interesting biological process for artificial learning algorithms [71; 69]. Hence, especially biologically plausible algorithms are often related to LTD and LTP. The 3rd generation of SNN is extremely powerful but requires more complex learning algorithms. The most common one is Spike-Timing-Dependent-Plasticity (STDP) [155; 141], which locally strengthens connections based on correlated activities as shown in Figure 2.11. When used for ANN, Hebb's principle defines how synaptic weights between artificial neu-

Learning	Pro	Con
Conversion	• ANN to SNN conversion is easily applicable • experience in training traditional networks can be used	• conversion causes performance loss • restrictions due to previous network architecture • spike timing neglected
STDP	• close to the biological model • different neuron models • spike timing incorporated • allows online learning	• usually applies the external non-spiking classifier • not as performant as ANN-training
Backprop	• very performant • spike timing is considered • SNN specific parameters are part of training	• adaptation of LIF model necessary • no scientific consensus about technique yet

Table 2.1.: Benefits and drawbacks of different learning methods for SNN.

rons are adapted. Thereby, if neurons are stimulated simultaneously, the weight of their connecting synapse increases and if they are activated separately it decreases. Hence, synapses are strengthened if a presynaptic spike occurs shortly before a post-synaptic one and respectively, weakened if a postsynaptic one happens right before a presynaptic spike. The learning rule STDP captures accurately this biological process and thereby increases the sensitivity of the post-synaptic neuron to presynaptic spike timing. It is often formalized as

$$\Delta w_{ij} \propto f(w) \times e^{\frac{-|\Delta t|}{\tau}}, \tag{2.6}$$

with the difference in time between the post- and presynaptic spike $\Delta t = y_i - y_j$, the time constant τ and the function f [87]. Besides the common form of STDP, Anti-Spike-Timing-Dependent-Plasticity (Anti-STDP), also called Anti-STDP, was observed in the human brain. Anti-STDP strengthens synapses in case a post-synaptic spike is emitted directly before a presynaptic one, hence, the synapses are strengthened in the opposite direction in contrast to standard STDP [155; 141; 156; 157] as visualized in Figure 2.11.

Learning with SNN is essentially a problem that has not yet been fully solved because there is no such general and performant solution as bb for deep learning. Therefore, some scientists resort to the method of training a conventional ANN and then covert it into an SNN. Thereby, the benefits of having a performant training algorithm are combined with the advantage of using energy-efficient hardware. The advantages and disadvantages of all three presented alternatives for training SNN are shown in Table 2.1. For an in-depth discussion about learning in SNN, refer to [147; 158; 159].

Parallel Hardware and Neuromorphic Chips

SNN are in theory very powerful, however, their simulation on Central Processing Unit (CPU), especially for large networks, is not performant [14]. CPU are based on the von Neumann architecture, thus, data and instructions are stored separately, resulting in sequential process execution [160]. SNN simulation is a parallel problem because three steps must be executed continuously for all neurons, *1)* neuron state update, *2)* synapse state update and *3)* weight adaptation, thus the actual learning [161]. Two parallel hardware architectures process SNN more efficiently than CPUs. First, neuromorphic hardware, specially developed for the simulation of SNN and based on Mead's analysis of processing in the brain [162] and secondly, the easily accessible and widely used Graphics Processing Unit (GPU).
The development of AI algorithms in recent years has been rapid and their success, especially regarding the application of neural networks and deep learning is highly impressive. To achieve this power, however, massive amounts of data and computing resources are necessary. The induced energy consumption is not sustainable and fit for the future. This problem is due to the enormous discrepancy between the processing methods of neural networks and the classical von Neumann architecture. Furthermore, biological brains are sophisticated regarding energy and data requirements, not only but in particular for tasks that are generally better solvable by humans than machines [163]. The research field *neuromorphic computing and engineering* aims to overcome the mentioned discrepancy caused by using sequential processors for parallel processes [163]. At its emergence the term *neuromorphic* referred solely to the artificial imitation of neurobiological structures and processes of the nervous system employing Very Large Scale Integration (VLSI). Today, the term is used more openly, describing a wide range of systems capable of simulating SNNs as well as resource-saving and biologically detectable algorithms and techniques regarding spiking neurons and their learning principles [163]. As shown in Figure 2.12, the motivation for neuromorphic engineering is two-fold. Firstly, to better understand neural processes through implementing physical emulations and secondly to conceive low-power innovative devices that strongly deviate from the conventional systems for sensory processing [163; 164]. Design decisions are hereby strongly influenced by the model given by the nervous system regarding not only neurons, synapses and network structures but also sensory systems. Even though neuromorphic

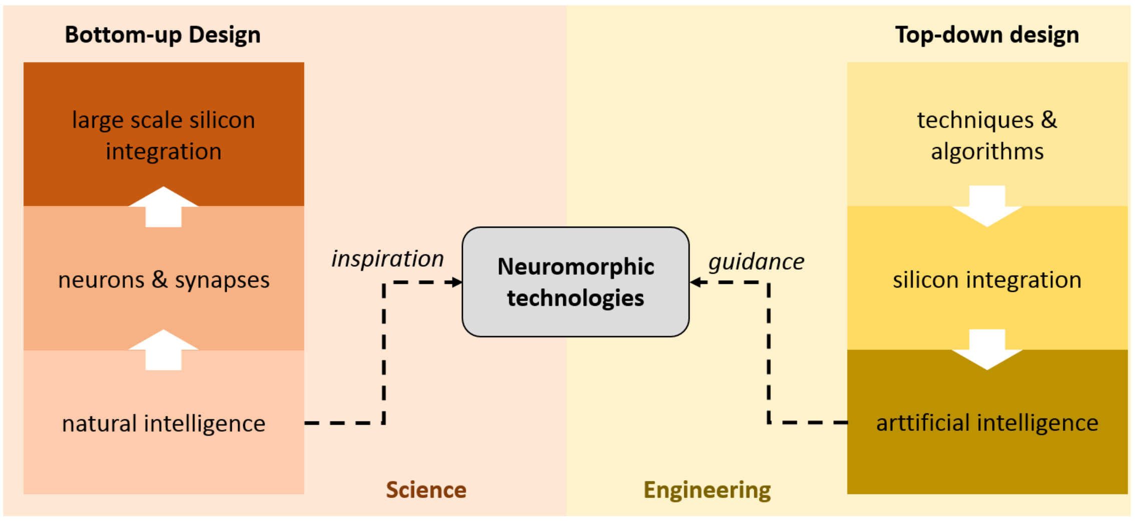


Figure 2.12.: Different perspectives on neuromorphic technologies. The bottom-up approach, replicating natural intelligence, is opposed to the top-down approach of applied research. Graphic is inspired by [164]

systems have a unique motivation and origin, in practice there are no precise definitions of what neuromorphic hardware must contain or achieve. However, processes and memory in a neuromorphic system tend to be analog and distributed while communication is usually time-dependent, direct and asynchronous [165]. There is a high-level distinction between neuromorphic systems whereby neurons are either implemented as digital or analog components, whereby analog is the more complex method [166]. The scalability of a neuromorphic system depends highly on its power consumption and space requirements. Hence, analog neurons are regarded as the more promising method as these are more efficient in both areas. A quantitative evaluation [166], comparing an analog implementation of a LIF neuron model with a digital one, showed that the digital implementation needs about 20 times more energy and 5 times more space. The use of analog boards is further supported by their ability to update continuous states without discretization errors [158]. However, as analog computations are less robust to noise and are temperature sensitive exact quantitative simulations and reproducible results are achieved better using digital circuitry [163]. Nevertheless, several analogs, as well as digital boards, have been developed in recent years. While basic design principles like Address Event Representation (AER) are shared, each board pursues different design goals [158]. Examples for analog neuromorphic boards are Neuromimetic ICs [167], Neurogrid [168] and BrainScaleS [169; 170; 171] and for digital ones TrueNorth [172], Loihi [11] and spiking neural network architecture (SpiNNaker) [10].
BrainScaleS's inter-chip communication is asynchronous [173]. It is designed to achieve maximum flexibility and runs 105 times faster than biological real-time, which makes it perfect for extensive experimental studies [158]. In contrast, Neurogrid runs in biological real-time, making it ideal for neurorobotics, and is optimized towards energy efficiency. However, this results in signifi-

cantly less configuration flexibility [158]. Regarding digital boards, Loihi and TrueNorth are both fully custom Application Specific Integrated Circuit (ASIC) realizations [174]. As TrueNorth was developed targeting commercial applications the implemented neuron and synapse models are relatively high-level and abstract [158]. Moreover, a system clock exists but is not used globally, which allows a partially asynchronous and partially synchronous device. Even though the system is deterministic, stochastic behavior is possible by using a pseudo-random source [173]. Unlike TrueNorth the Loihi design is mostly asynchronous. It aims at the integration of learning rules which are distributed over the whole network to allow more efficient learning without transporting data over long routes. Furthermore, this chip has no of-chip memory, thus computation and storage are integrated. In strong contrast to the systems presented so far, SpiNNaker uses standard Advanced RISC Machines (ARM) microprocessors to manage neural dynamics. On the one hand, it enables more flexibility as well for neuron models as network topologies [158]. On the other hand, it leads to a less efficient architecture, as the control memory and unit become an overhead [175]. SpiNNaker uses a hierarchical implementation of AER as the communication framework allowing for great scalability regarding network size [173]. Comprehensive overviews about the design and applicability of neuromorphic hardware are given in [158; 173; 174].
GPUs are well suited for parallel computations, but not per se suitable for sparse communication, as necessary for simulating SNN. To overcome this issue, and thereby harness the computational power of a parallel hardware architecture that is more accessible and cheaper than neuromorphic hardware GPU-enhanced Neuronal Networks (GeNN) was designed [12]. It is a code-generating C++ library enabling users to run neuron models and learning rules efficiently on GPU, without having expert knowledge of low-level Compute Unified Device Architecture (CUDA) programming.

2.2.2. Self-organizing Neural Networks

A popular algorithm to create a spatially organized representation of large quantities of unstructured data to disclose correlations between data items is the Self-organizing Neural Network (SONN) and especially its most famous representative the Self-Organizing Map (SOM) [97]. In contrast to ANN, or related ML methods, SONN are unsupervised. SONN provide an efficient way to reduce the dimensions of presented high dimensional data while preserving similarity relations and inducing a topological structure and parameterization [176; 177]. Even though many deviations and extensions of the standard SOM exist, the most common is a 2D, non-linear approximation of a high-dimensional data manifold [97]. Ideally, the mapping should function in such a way that all instances of the input space are represented by a smaller number of neurons so that similar input data is represented by neurons that are close to each other [176]. The SOM's success story is largely based on the straightforward definition and relatively simple practical handling. Its interesting clustering properties and visualization abilities led to a wide range of application possibilities in the most diverse fields [91]. An important cornerstone for SONN was laid in 1980 by the Finish Professor Tuevo Kohonen [178] who invented the SOM. The SOM is an extension of the k-means algorithm, whose peculiarity is that it preserves well topological structures of the data [91]. Historically, the formation of topographic maps began with two types of self-organizing processes as well as two types of network architectures [97]. Besides the well-known *competitive learning*, used in many SOM related algorithms, *gradient-based learning* forms an alternative learning process for self-organization. Regarding network architectures, the *Willshaw-von-der-Malsburg model* [94] consists of two sets of neurons, the input space is therefore also discretized. The Kohonen model [178] includes only one set of neurons, the output space, while the input space is continuously valued. Nowadays SOM apply almost exclusively competitive learning and the Kohonen architecture. In the early eighties, SOM were primarily used for neuro-biology modeling [91]. However, this changed drastically after a short time. In [179] a variety of the algorithm's application areas are listed, emphasizing its versatility despite the actual simplicity of the method. This includes but is not limited to, vision, speech and language [180; 181], musical studies, process control, robotic motion control [182], design of electronic circuits, chemistry [183; 184; 185], medicine, biology, economics and mathematics. A comprehensive compilation of the various application areas has been renewed several times [186; 187; 188]. A SOM has also been used for stereo vision, more precisely to solve the correspondence problem, thus to match the corresponding dots of two images from the same scene. Early approaches are presented in [189; 190], while [191] uses self-organization for 3D hand pose estimation and [192] generates 3D face models from stereo images. Comprehensive overviews, each including the latest developments at the time, have been proposed in [97; 177]. A review regarding the application of SOM for robot motion control is given in [182]. A big step towards the biological model, introduced in section 2.1, was made in [96; 193; 194] by implementing self-organizing networks with spiking neurons (see subsection 2.2.1).

Learning through Adaptation

The learning process is divided into two phases [97]. Firstly, within the *competitive stage*, the winning neuron, referred to as the Best Matching Unit (BMU), is selected. This is achieved by a "Winner-Takes-All"-approach, also referred to as lateral inhibition, thus only one neuron is fired at any time. Therefore, each neuron $a \in A$ of the output layer competes to determine the neuron whose synaptic weight has the smallest Euclidean distance d to the input x. This can be formalized as:

$$BMU = \min_{\forall a \in A} d(x, w_a) \tag{2.7}$$

By use of the Euclidean distance, a Voronoi diagram of the input space is obtained. Thereby each neuron of the map represents an input space region, bounded by perpendicular bisectors connecting pairs of weight vectors. Secondly, in the *cooperative stage* the BMU synaptic weights, and to a lesser extent also those of its directly neighboring units, are updated. After the presentation of each input vector, the network's weight vectors are updated by [195]:

$$\Delta w_a(t+1) = \eta(t) \cdot \theta(t)||x(t) - w_a(t)||. \tag{2.8}$$

Thereby, the continuously decreasing learning rate is given by

$$\eta = \eta_0^{\frac{i}{\#samples}} \tag{2.9}$$

and $\theta(t)$ is the SOM's neighborhood function, which defines the BMU's shrinking radius. Equation 2.8 adapts the weight vectors of the BMU and topologically-related units, as it is centered at the BMU and its impact decreases with distance. The neighborhood function θ for each neuron a is thus computed for every learning cycle by

$$\theta(a) = exp^{-\frac{\delta(a,a_{BMU})}{2\sigma^2}} \tag{2.10}$$

by which, neurons located topologically close to each other learn to respond to similar inputs [97]. The extent to which the radius of influence on neighboring neurons decreases is given by the neighborhood rate

$$\sigma = \sigma_0^{\frac{i}{\#samples}}. \tag{2.11}$$

Due to the drop regarding the learning rate η and neighborhood rate σ, the influence of new training samples falls drastically during learning. This allows strong exploitation in early iterations and exploration at a later stage.

The distance between a neuron a and the current BMU is given by $\delta(a, a_{BMU})$. This value is meant topographically, in contrast to the weight distance in Equation 2.13. How a learning step impacts the BMU, colored in red, and its less colored topological neighbors, is visualized in Figure 2.13(a). The graphic shows the difference in the output space before and after a learning step.

Weak Points and Further Developments

Since its invention, the SOM algorithm has been used in many different fields and with many different extensions. According to [196], this is because *"its implementation on a computer is straightforward and numerically robust"* and in [197] it is stated that it is *"very easy to define and to use, and a lot of practical studies confirm that it works"*. Also, it is a major speedup compared to conventional algorithms using one thread and additionally, it does not require any dedicated hardware [198]. Even though its many beneficial and interesting features made the SOM a popular algorithm in many fields, there are disadvantages worth mentioning. Many of these drawbacks are encountered by specific versions of SONN, however, there is no perfect solution that solves all problems [176]. In Table 2.2 some of these weaknesses are listed. However, it should be noted that some of the weaknesses

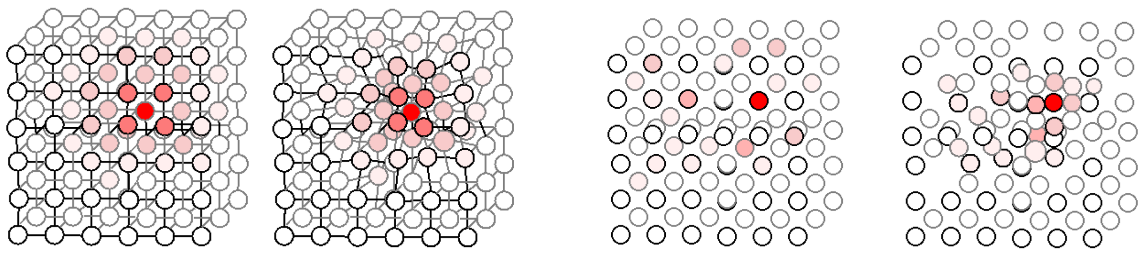


Figure 2.13.: (a) features the output space of the SOM and (b) of the NG. The left part of each image shows the network before, while the right part displays it after learning. The BMU is colored red and its neighbors are colored in a graduated hue, depending on their distance. However, distance is defined topologically in (a), thus, it is noticeable that neighboring neurons are pulled towards the BMU. In contrast to that, the similarity of the neurons' weight vector to the input vector is crucial in (b). Hence, the colored neurons adapting to the BMU are not necessarily neighbors in a topological manner. Image source: (Steffen et al. 2021c; Glueck 2021)

are only defined as such for certain applications and are not necessarily general disadvantages. The original version of the SOM, the online SOM, is not deterministic, thus diverging maps might occur in different runs [91]. Depending on the target application, however, this property may have disadvantages. To overcome this problem, Kohonen himself introduced the Batch SOM [199] which gets reproducible results with the same initialization. The downside of this version is that it neglects the observation that not all matches are based on a significant association of its pair. Meaning that if two data points are within the same cluster it might be due to being closely located in the input space but it might also be coincidental [91]. As the BMU search is quite complicated and time-consuming in [200] "boosted" versions are introduced, which decreases the training time. This is achieved by a hierarchical approach and sub-optimal solutions of the nearest neighbor problem [176].
Due to its winner-takes-all approach, applied by the competitive learning algorithm, neurons are only adapted if they are the selected BMU in a certain learning step. Even if the weights of several neurons are very similar, and the BMU is only determined by a slim margin, the second and third-best neurons are not adapted regarding the input vector. Local learning may lead to local minima [210] and is not very performant. Thus, many researchers developed methods to speed up the original SOM [210; 211; 212]. The NG [201] was introduced by Martinetz and Schulten and its name is inspired by the gaseous way it covers its regions. It differs from the SOM fundamentally, as it has no rigid structure, as visualized in Figure 2.13(b). Thus, learning is not local and the neurons that are adapted beside the BMU are chosen regarding their weight vectors' similarity to the input vector. For NG, competitive learning is thereby applied to all neurons, instead of the SOM's winner-takes-all approach, as shown in Figure 2.13. Therefore, for the NG

Issue	Solutions
visualization of clusters	Kohonen Maps [178]
non-deterministic [91]	Batch Self-organizing Map (Batch SOM) [199]
complex BMU search [176]	"boosted" versions [200]
winner-takes-all-approach	Neural Gas [201]
topological mismatch [202]	Topology Representing Network [202]
complex previous sizing [176]	Growing Neural Gas [203]
time-series analysis [204; 182]	Merge Growing Neural Gas [205]
spatiotemporal quantization	Segment-GNG [206; 207]
relative distances not visible [208]	Visualization Induced SOM [209]

Table 2.2.: Overview of issues regarding SONN and subforms that address them.

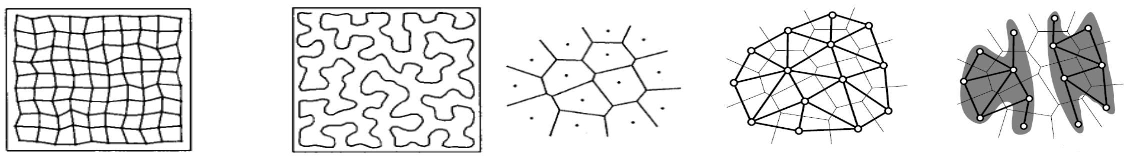
the weight update is like for the SOM as given in Equation 2.8. What changes, however, is the neighborhood function from Equation 2.10 that becomes

$$\theta(a) = exp^{-\frac{\delta(rank(a))}{2\sigma^2}}. \tag{2.12}$$

The topographical distance is replaced by a ranking system, based on the neurons' weight distance to the training sample x, given by

$$d(x, w_a) = ||x - w_a||. \tag{2.13}$$

Hence, a neuron a is more strongly adapted the higher its value $\theta(a)$ is, as it indicates a high similarity of the weight vector of a to the training sample x.
Techniques for dimensionality reduction often struggle with a high dimensional space holding low dimensional data structures that are non-linearly embedded [208]. SOMs preserve the topology of the input data during learning, thus, adjacent neurons in the output space are also neighbored in the input space [178]. However, neighboring neurons in the input space are not necessarily adjacent in the output space. The only exception is if the topological structures of input and output space match each other [202]. Only if the mapping between a manifold M and graph G is neighborhood preserving in both directions, G forms a topology-preserving map of M. And only then, do adjacent neurons on G correspond to features neighboring on M and vice versa [202]. In Figure 2.14(a) & (b) graphs are depicted of the 2D manifold M which is formed by a SOM. In (a) the graph $G1$ is 2D, it forms a lattice with the same topology as M. Neighboring nodes of $G1$ are also neighbored by M and vice versa, thus, $G1$ is topology-preserving. In contrast the 1D-graph $G2$ in (b) forms a string, therefore, adjacent neurons on $G2$ are also neighbored on M but the opposite does not hold. As the mapping from $G2$ to M preserves the topology but not from M to $G2$, $G2$ is folded in M. Hence, this network has good quantization but no topological preservation. The NG [201] also quantizes a given manifold but it does not provide a



(a) topological match (b) topological missmatch (c) Voronoi Diagram (d) Delaunay Tri. (e) Induced Delaunay

Figure 2.14.: (a) & (b) show graphs of the 2D-manifold M formed by Kohonen's SOM algorithm. As graph $G1$ in (a) is also 2D, it forms a lattice with the same topology as M. In contrast the 1D graph $G2$ in (b) forms a string. Both graphs have a good quantization of M, but only $G1$ can achieve a topology-preserving mapping. Sources: (a) & (b) from [202], (c) from [201], (d) & (e) from [203]

defined topology between the units. Thus, the developer of NG, Martinetz and Schulten, introduced Topology Representing Network (TRN) [202]. Hereby, vector quantization of the NG algorithm is complemented by a competitive Hebbian learning rule forming a synaptic structure. The Hebbian rule [82], stating that repetitive stimulation strengthens a synaptic link, is also a foundation for learning with spiking neurons [155] as introduced in subsection 2.2.1. Therefore, the TRN is remotely related to SNNs. In literature, the term "topology representing networks" also refers to a group of SONN, able to generate an optimal topology-preserving map for diverse data structures [208]. This also includes Dynamic Topology Representing Network (DTRN) [213], an incrementally growing TRN and Weighted Incremental Representing Network (WINN) [214]. The algorithms TRN, DTRN and WINN are very alike, however, the most robust data representation is achieved by the TRN [208]. A major inconvenience in practice is that the appropriate configuration of the map must be determined by intelligent guessing and testing. Thus, a suitable lattice setup and size are usually not easy to determine and can only be determined by test runs of the network. However, it is very important for a good performance and convergence of the map, as a wrong configuration or lattice size may lead to falsely represented data [176; 215; 216]. An incremental version of NG, capable of detecting the required grid size during learning, was proposed by Fritzke [203; 217] as the Growing Neural Gas (GNG). The network is an undirected graph that does not have rigid connections and an unrestricted topology [218]. It is initialized with two neurons and grows itself by adding nodes incrementally using competitive Hebbian learning [219]. The graph is altered by adding and deleting neurons and synapses to minimize an error function [218]. Thereby, every node has a local error which is only updated if the neuron is chosen as the BMU. New nodes are inserted in areas with a high accumulated error.

Time plays an important role in input data like robotic sensor streams as well as medical recordings like Electroencephalography and Magnetoencephalography or for speech recognition. Thus, considering temporal structures while learning is crucial. The original SOM, like most unsupervised learning models, only considers vectorial data, disregarding temporal structures altogether, therefore unable to use the powerful tool of self-organization for data with temporal characteristics [204]. In other words, the SOM performs a static mapping, which is unsuited for real-world data as it is often dynamic [182]. The authors of [182] even claim that processing spatiotemporal patterns is among the most important features of any intelligent system. In [204] a review of SOM for temporal structures is provided. Additionally, a very comprehensive article on the subject, which is supplemented with a table of all SOM models on time sequences, is given in [182]. SONN models for clustering and analyzing large time series can be roughly put into five categories [204; 97]:

1. fixed-length windows [220; 221; 222]
2. specific sequence metrics [179; 223]
3. statistical modeling, integrating generative models for sequences [224; 225]
4. mapping of temporal dependencies to spatial correlation [226; 227; 228]
5. integration of previous states
    a) own past activation [229; 230]
    b) past states of other neurons [231; 232; 233; 234; 235]

The 5th category, SONN which integrate previous activation, includes for *(5a)* the Temporal Kohonen Map (TKM) [229] and the Recurrent Self-organizing Map (RSOM) [230]. Both extend the SOM architecture by recurrent synapses allowing one to take a neuron's previous states into account, for calculating the BMU. Hence, TKM and RSOM use temporal context implicitly, meaning they do not consider previous learning progress but instead previous training samples. This technique uses fractal encoding and is also referred to as *Cantor sets*. The TKM is more closely related to the biological model. Thus, it increases the chance of a previous BMU to be selected again by adding a decaying activation potential to each winning neuron. The working principle of the RSOM is more simple, thereby the previous input vector is used in each learning cycle in addition to the current training sample. In contrast to that the Recursive Self-organizing Map (RecSOM) [231] and the SOM for structured data (SOMSD) [232] in *(5b)* integrate past states of other neurons. This is also the case for the Merge Self-organizing Map (MSOM) [233], its extension the γ Self-organizing Map (γ-SOM) [234] and the respective extension of the GNG, the γ Growing Neural Gas (γ-GNG) [235]. Hence, these models have an explicit temporal context by applying an additional context vector, which integrates previous learning progress instead of old training samples. The recurrent feedback connections of the RecSOM combine the current input vector with a previous state of the network. Therefore, each neuron is equipped, besides the weight vector, with a context vector representing temporal context as past network activation. However, as the context vectors' dimension is the map's number of neurons, this form of temporal context is very computationally expensive. This problem, however, does not exist with the more compact MSOM, whereby the context vector, that each neuron has in addition to

the weight vector, is of the same dimension as the weight vector. Here the context vector is a linear combination of a neuron's current weight vector and the context vector of the last BMU. As an interesting side note, the weight representation of the MSOM, TKM and RSOM is very similar, however, due to the explicit context vector, the MSOM is more stable.
A recent extension of the GNG is the Segment Growing Neural Gas (SGNG) [206; 207]. It is a growing network model which also takes temporal context into account. However, it differs greatly from all versions introduced, as its input is not joint angles, but instead, parts of the trajectory, referred to as segments. This enables a whole new approach to spatiotemporal quantization. As SOM map the input vector on an ordered grid, local distance information between data points on the map are not visualized [208]. The Visualization Induced SOM (ViSOM) [209] extends the SOM to preserve the topology of data as well as inter-point distances. This information about relative distances between data points allows an effective visualization of how data is structured and distributed [208]. In Table 2.3 the ad-

Version	Pro	Con
SOM	Simplicity, topology-preserving [178], no adaptation parameter to tune	Static learning & dimension, many iterations necessary, a priori knowledge needed to predefine size and parameters [215; 216; 178; 176], topological mismatch [202]
Neural Gas	dynamic learning parameters, continuous training, good for depicting topological relations [236]	strong dependence on initially set lattice size [216], static dimension [215]
GNG	dynamic dimension [215], low computation time [205], continuous error reduction [215], good adaptation to data structure	small number of initial neurons may cause information loss [205], very sensitive to parameter setting [216]
MGNG	recursive temporal dynamics, very performant [205]	bad adaptation for small changes, input data cannot be randomized, information loss in the beginning

Table 2.3.: Advantages and disadvantages of different SONN. The original SOM [178], NG [201], the iterative GNG [203], and a model with temporal context, the MGNG [205].

vantages and disadvantages of different SONN versions are summarized. For clarity, the table is limited to 4 versions, the original SOM [178] and NG [201] are complemented by a growing network, the GNG [203], and a model with temporal context, the Merge Growing Neural Gas (MGNG) [205]. The original SOM impresses by simplicity and its ability to preserve topology [178]. However, due to rigid interconnections between adjacent neurons, learning is static and connections may exist between unrelated nodes [178]. Also, many iterations are necessary for learning due to a separate train and test phase. A priori knowledge of the data is crucial as a bad choice of the number of neurons makes it lose capacity [216]. Connections cannot be removed or added during learning [178], therefore, the number of neighbors [216] must also be set initially. To predefine lattice size and parameters, the number of clusters should be estimated beforehand [215]. In contrast to that, for the NG prior knowledge of the data structure is not as crucial. This model can form node clusters, delete connections between unrelated nodes and adapt neighboring nodes flexibly [236]. Hence, the NG fits various topologies and is well suited for depicting topological relation [236]. Furthermore, training can be continued during test phases, which speeds up the process significantly [236]. However, its performance is still highly dependent on the initially set lattice size as the number of neurons is fixed [216]. Growing networks [203] cannot only add and delete connections but also neurons flexibly where needed. Hence, nodes and edges are pruned continuously and the amount of neighboring neurons is flexible [216]. This enables them to adapt well to any given data structure without prior knowledge [236], enhances continuous error reduction and also allows a dynamic dimension [215]. GNG are very popular due to their ability to dynamically adjust to any data structure and a relatively simple network structure [236]. Finally, these networks show a lower time complexity than SOM or NG, causing a run time advantage and better computation times [205]. However, as GNG are very small at the start of the training phase they are not able to represent data yet which may cause information loss [205]. Also, these networks are very sensitive to training parameters, thus, a good setting is crucial [216]. The last model of Table 2.3, MGNG, introduces recursive temporal dynamic and was developed specifically for time series analysis [205]. Hereby, subgraphs can be seen as clusters. MGNG include all benefits of GNG and are even more performant. As with GNG, there is also a risk of the initial loss of information. Furthermore, input data cannot be randomized. Finally, if the differences between succeeding steps are small, the context vector will not be able to represent the difference.

Quality Measure

As training a SONN is an unsupervised learning process, it cannot be evaluated against ground truth, unlike supervised learning methods. To evaluate how well a SONN performs clustering, a quantitative evaluation can be done [237]. It is important to choose an indicator that also considers the topology of the map. Two much-used metrics are the Quantization Error (QE) and the C-Measure (CM). The QE sums up the differences of all N training vectors x and their determined BMU, which can be formalized to

$$QE = \sum_{i=1}^{N} ||x_i - BMU_i||^2 \tag{2.14}$$

as adapted from [237; 199; 238]. A good input space approximation is thereby indicated by a low QE.
The CM was first introduced in [239] and has since become a standard for evaluation of SONNs [237]. This value provides information about how well the neighborhood structures between neurons are maintained from the input to the output space. To calculate the CM, the element-wise products between pairwise distances of input vectors x and their BMUs are summed up, as formalized by

$$CM = \sum_{i=1}^{N} \sum_{j<i} d(x_i - x_j)\delta(BMU_i, BMU_j). \tag{2.15}$$

For a comprehensive overview of quality measures for SONNs see [237].

2.2.3. Event Cameras

As frame-based cameras existed longer than computer vision, up until recently vision algorithms did not only use but, heavily rely on this synchronous mode of action, which has some adverse effects on performance and accuracy. Responsi-

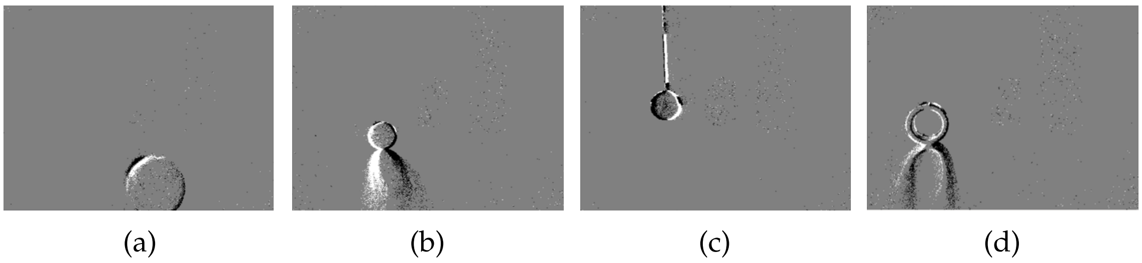


Figure 2.15.: Event frames of fast moving objects. A ball moving (a) towards and (b) in parallel to the sensor. (c) A pendulum and a (d) rolling cylinder, also moving in parallel to the sensor. Each white dot is an ON-event and each black dot is an OFF-event. Image source: (Ehrlinspiel 2019)

ble for this are characteristics of conventional cameras such as high latency, low dynamic range, and – to some extent – high power consumption. The classic example of what frame-based sensors are not suitable for is high-speed movements. The inevitable motion blur either prevents image processing, severely impairs accuracy, or requires computationally intensive post-processing [240]. From a hardware point of view, there is a need to increase the frame rate, however, this increases energy consumption and leads to enormous amounts of redundant data due to synchronous image generation. In contrast, as shown in Figure 2.15, event cameras' asynchronous mode of operation allows an extremely high temporal resolution for dynamically changing scenes. At the same time, no unnecessary data is generated, for static scenes. Thereby, Figure 2.15 shows nicely that the static scene is completely ignored and only the moving part is perceived by the event camera. Thus, these sensors solve both the under and over-sampling problem, with extremely low power consumption [241]. Event cameras, or silicon retinas, are inspired by the visual system and, thus, modeled on the retina (see Figure 2.5). These sensors technically implement neurobiological principles, as introduced in subsection 2.1.3. By asynchronously measuring brightness changes for individual pixels, these sensors differ greatly from conventional cameras which transmit data synchronously as frames [15]. A very early model of a silicon retina

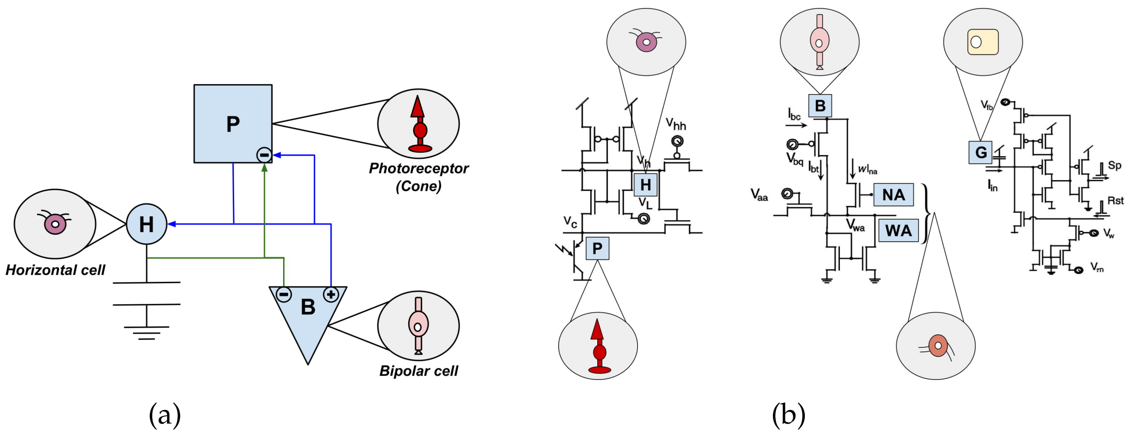


Figure 2.16.: Artificial building blocks and their biological counterparts of early prototypes for Event Cameras. (a) The silicon retina designed by Mahowald and Mead embodies three components inspired by photoreceptor, bipolar and horizontal cells. (b) Zaghloul and Boahen's prototype additionally models ganglion and amacrine cells. The left circuit models the outer retinal layer with the photoreceptor (P) and horizontal cells (H). The amacrine cell modulation is represented by the central circuit. Wide-field (WA) and narrow-field (NA) amacrine cells are stimulated by a bipolar terminal (B) while simultaneously inhibiting B. The circuit at the right embodies ganglion cells (G) as a membrane capacitor which can emit spikes and is subsequently discharged. Image source: (Steffen et al. 2019c)

was developed at Caltech by Mahowald and Mead [242; 243] in 1991. This prototype, including the biological association of its components, is shown in Figure 2.16(a). Even though Mahowald's sensor [242] was exclusively applied for test and demonstration purposes, event cameras, developed in the last 30 years, are based upon its conceptual architecture [244]. The Parvo-Magno Retina from Zaghloul and Boahen [113; 245] imitated biological principles even more realistically, as shown in Figure 2.16(b), but was not superior to Mahowald's model in terms of application technology. In contrast, the developments of Ruedi and

Mallik [246; 247] were convincing due to their technical maturity, however, they used a synchronous mode of action, which strongly limits the biological plausibility.
Instead of generating images at a fixed rate, event cameras output event streams. For a digital representation events are formalized as tuples, $Event(x, y, t, p)$, with x and y referencing the pixel location, t the timestamp and p the polarity or sign of the brightness change. The polarity indicates whether the lighting intensity has increased, for an ON-event, or decreased for an OFF-event. In Figure 2.15 each white dot is an ON-event and each black dot an OFF-event, as the recordings are carried out on an Asynchronous Time-Based Image Sensor (ATIS). The colors show that (a) is moving towards the viewer, while the direction of movement in (b) and (c) is from left to right and in (d) from right to left. Biological data

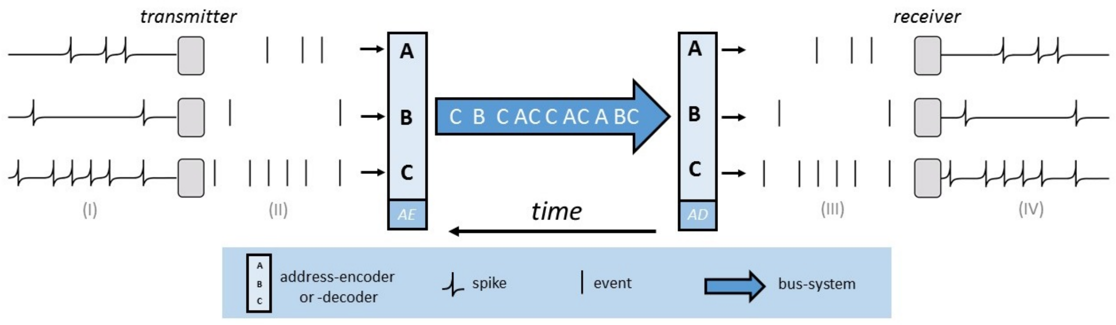


Figure 2.17.: Simplified schematic drawing of the AER-bus-system. (I) Neurons of the sending chip generate several spikes, (II) which are interpreted as events for which the address encoder (AE) generates a binary address. (III) The bus system transmits the address which is subsequently decoded by the receiving chip's address decoder (AD). (IV) The decoded event address triggers a spike of the respective neuron. Image source: (Steffen et al. 2019c)

transfer from the eyes to the respective neural regions involves one million axons of 366 ganglion cells. A realistic artificial approach would require a separate cable for each pixel [106]. A lightweight form to transmit the digital representation of events is the AER protocol, which was initially introduced in [248]. It is a hardware protocol widely used for spike-based and event-based communication up until today. A simplified visualization of the AER-bus-system is depicted in Figure 2.17. The graphic shows how three neurons of the sending chip generate spikes which are encoded by the address encoder, transmitted via the AER-bus, decoded by the address decoder and finally occur on the receiving chip. While the polarity is encoded with one bit (OFF/ON) and the timestamp usually by 32 bits, how many bits are used for the event's address is dependent on the resolution of the event camera [240].
Recently, event cameras have become quite popular, not only in academic circles but also in commercial spaces. This is mainly because their performance is continuously increasing and frame-based sensors have been exponentially over-

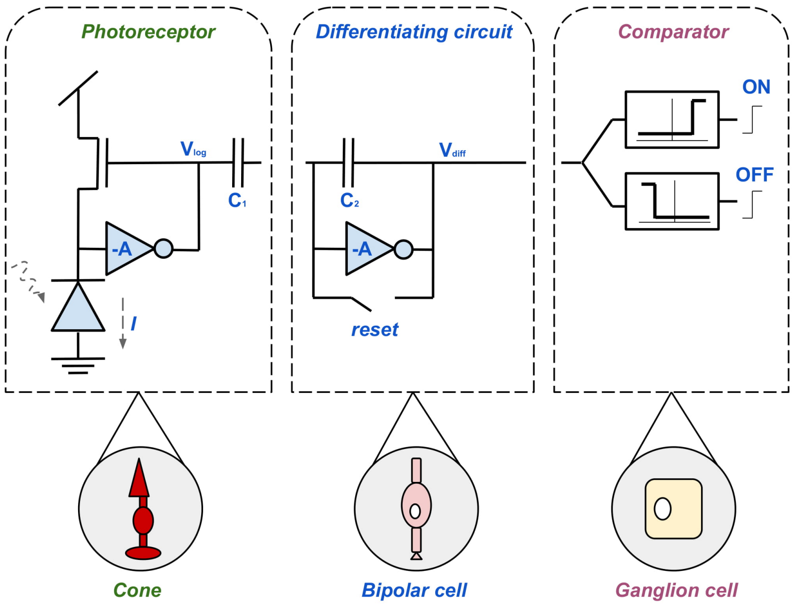


Figure 2.18.: Technical pixel circuit of the DVS divided into three parts which are assigned to their biological model. The photoreceptor, the light-sensitive component, models biological cones, while the differential circuit is based on bipolar cells and the comparator on ganglion cells. Image source: (Steffen et al. 2019c)

taken in terms of speed [240]. Compared to frame-based visual sensors, event cameras have several attractive properties, especially for application in robotics or computer vision tasks [15]:

- high temporal resolution in the order of µs
- very high dynamic range ca. 140 dB
- low energy demand
- high pixel bandwidth, on the order of kHz

These advantages stem from three fundamental design decisions, inspired by the biological vision, of these sensors: First sparse event streams, second encoding luminance changes, third signal rectification and third the separate display of positive and negative illumination changes by OFF- and ON-signals. The biggest difference this makes for event-based sensors compared to conventional cameras is that imaging is no longer dependent on an external clock but takes place autonomously and individually at the pixel level [106]. The first practical and commercially available event camera, Dynamic Vision Sensor (DVS) presented in [244], was developed by a team of researchers of ETH Zürich. As shown in Figure 2.18, its pixel circuit replicates an abstraction of three components of the biological retina, creating a photoreceptor-bipolar-ganglion cell information flow [106]. This mechanism enables the sensor to be sensitive to the scene dynamics, thus, responding to each pixel in real-time to changes. Unlike the prototypes of Mahowald and its team, shown in Figure 2.16(a), the DVS has modeled the comparator on the ganglion, not the horizontal cells. Compared to standard frame-based cameras, the DVS suffers from a relatively big pixel size of 40 µm and a low resolution of 128×128 pixels. However, technical development made great strides since the release of the DVS in 2008, which can be seen by examples like Dynamic and Active-pixel Vision Sensor (DAVIS)240 [249; 250], ATIS [251; 252] and Samsung DVS-Gen4 [253]. For technical details refer to Table 2.4 and for a comprehensive list to table I in [15]. It is noticeable that researchers and man-

model	**resolution**	**pixel size**	**power consumption**	**year**
DVS [244]	128×128	40 µm	23 mW	2008
DAVIS [250]	240×180	18.5 µm	5 - 14 mW	2014
ATIS [252]	480×360	20 µm	25 - 87 mW	2017
Samsung DVS [253]	1280×960	4.95 µm	130 mW	2020

Table 2.4.: Overview of the technical development of event cameras over the years. Thereby well-known representatives were chosen that went hand in hand with major innovations. The DVS refers to the first commercially available event camera, DVS128. Additionally, the DAVIS240, Gen3 ATIS and Samsung DVS-Gen4 are presented. For a more comprehensive table see [15].

ufacturers focus particularly on the improvement of certain characteristics. Not surprisingly, spatial resolution plays a major role, which was often mentioned as a major disadvantage in the DVS. Nevertheless, even the event camera with the largest pixel array [253], as of today, has a spatial resolution of only 1Mpixel (1280×960). The low spatial resolution is of course a limitation in terms of applications. However, increased readout speed and since recently reduced pixel size are heavily targeted [15]. In particular, pixel size is an interesting problem as event cameras have a mixed-signal circuit. As can be seen in Table 2.4, the most common event cameras, DVS (40 µm), DAVIS (18.5 µm) and ATIS (20 µm), have large pixel sizes compared to standard conventional Active Pixel Sensor (APS) with about 2-4 µm. However, the table also shows that event cameras are far ahead of frame-based cameras in terms of power consumption. This is possible because no redundant data is transferred, instead only brightness changes on pixel level [15]. A significant reduction of pixel size will most likely only be possible if the purely asynchronous circuit design is abandoned in return [254]. Some manufacturers also provide new features as gray level output (DAVIS, ATIS), Inertial Measurement Unit (IMU) integration [255] and synchronization between cameras [256]. The pixel design of the DAVIS [249; 250] is based on the DVS (see Figure 2.18)), however, the circuit responsible for Change Detection (CD), generating event streams, is extended by an APS [257], as shown in Figure 2.19.

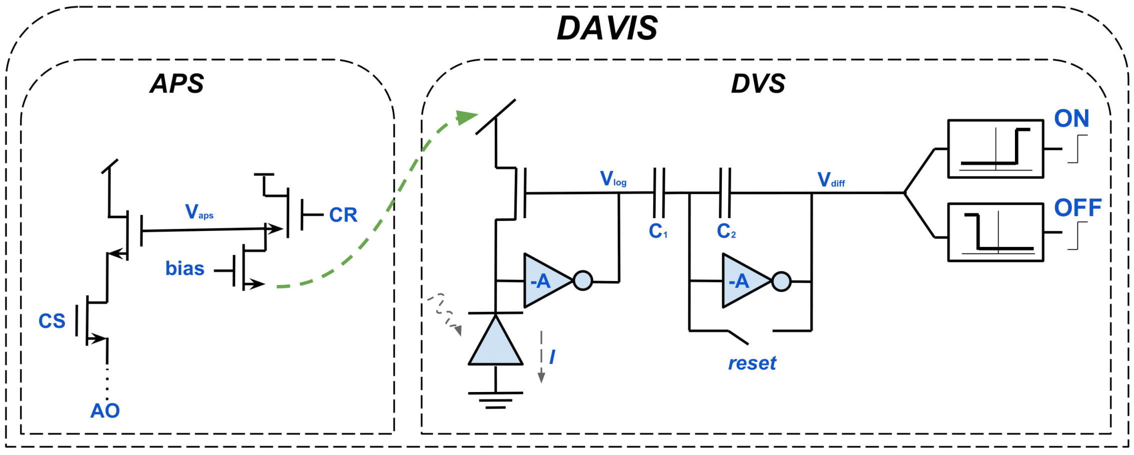


Figure 2.19.: Circuit of an DAVIS-pixel extending the asynchronous generation of event streams with synchronously generated gray level images using an APS. Thereby the DAVIS combines frame-based and event-based data acquisition on the pixel level. Image source: (Steffen et al. 2019c)

As this extension is done on a pixel level, the frame-based APS and the CD circuit share a photodiode for each pixel. APS capture the scene's illumination, thereby generating gray-level images as frames. Thus, DAVIS combines properties of frame-based with event cameras on a pixel level processing information both synchronously and asynchronously.

Like for the DAVIS, each pixel of the ATIS includes two circuits, the basic circuit of the DVS, the CD triggering event streams and Exposure Measurement (EM) generating grayscale images. However, very different from the DAVIS, both circuits operate completely asynchronously and are biologically inspired. While the CD, inspired by the magnocellular pathways, is responsible for the *"where"* ((this is also true for DVS and DAVIS), the EM, inspired by the parvocellular pathways, is responsible for the *"what"*. For the ATIS grayscale information is also transmitted via AER and consists of two events, EM integration start and EM integration end event [258]. Thus, grayscale information is reconstructed using luminance information which is calculated as the inverse of the time difference between those events:

$$I = \frac{1}{t_{e^+} - t_{e^-}} \tag{2.16}$$

As visualized in Figure 2.20, the circuits for CD and EM within the ATIS are mostly detached, and therefore have separate output channels. The only interaction is that the CD triggers the EM circuit. Thus, whenever the CD circuit emits an event, the EM circuit starts to accumulate incoming light until the accumulator is saturated. Also, if another event is registered for the respective pixel before the integration is finished, the CD circuit triggers an abort and restart of EM, as the previous EM is already obsolete [251]. This also implies that if the frequency

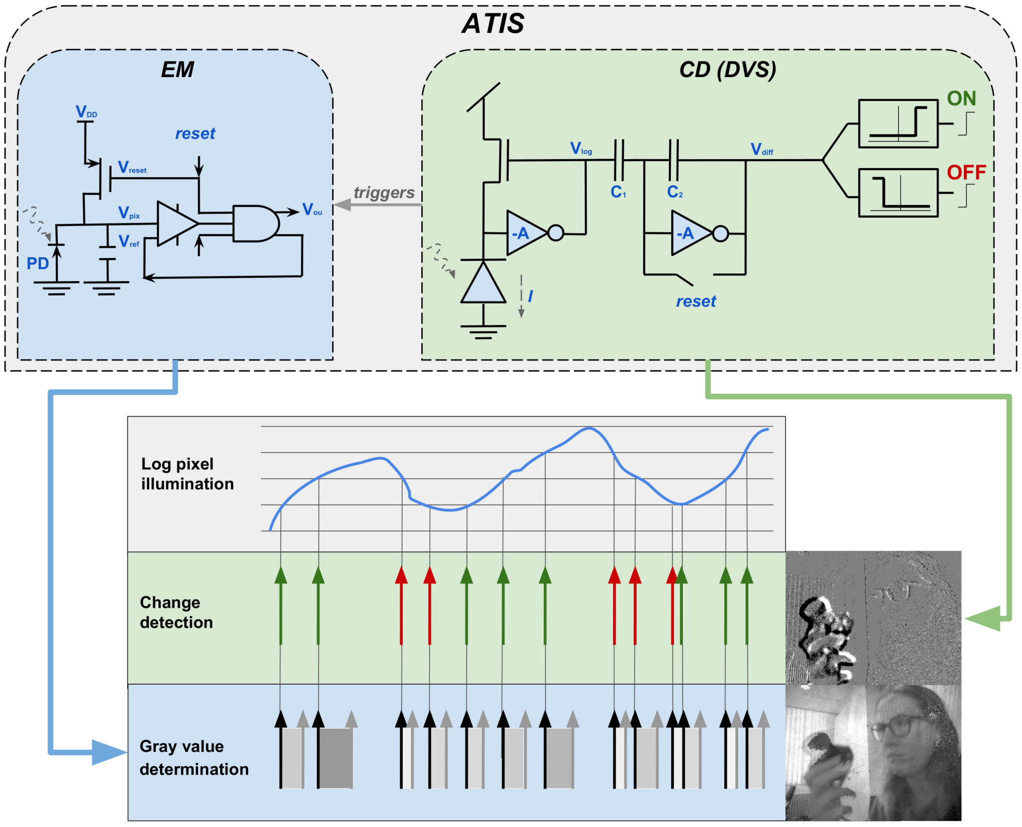


Figure 2.20.: Circuit of an ATIS-pixel, extending the basic circuit for CD to create event streams with a second circuit for EM to obtain grayscale images with entirely asynchronous image acquisition. The upper part shows that CD & EM are detached circuits, however, CD triggers EM to ensure that a pixel's gray value is updated for each event. The lower part visualizes how gray values are determined by Equation 2.16. Thus, small temporal differences imply brighter grayscale values than big differences. Image source: (Steffen et al. 2019c)

of events is higher than the EM integration time, involved pixels will not generate grayscale values. However, a frame-based camera, or the APS circuitry of the DAVIS, could not produce usable images under these conditions. In terms of biological plausibility, the ATIS is ahead of the other representatives due to its completely asynchronous pixel design. It is also very good in terms of temporal resolution and dynamic range. However, the asynchronous operating principle leads to uneven exposure times, causing motion artifacts and issues with slow, close objects [259]. An advantage results from the synchronous design principle regarding the pixel size as the photodiode is shared for the DAVIS. However, APS limits the dynamic range to 55 dB and generates redundant data [15]. For a comprehensive description of the biological and technical aspects of event cameras, especially DVS, DAVIS and ATIS, see [106; Steffen et al. 2019c].

3. Event-based Stereo Vision

For human stereo vision, all the depth stimuli presented in section 2.1.3 are used in combination. However, most of the sub-types depicted in Figure 2.6(a) only lead to relative values and are not very potent for themselves, thus they are more of a supportive nature. By far the strongest depth criterion, due to producing absolute values and providing relatively high accuracy, is binocular perception [126; 120; 109]. As it is extremely complex to implement all or even only a few of the depth criteria in a meaningful combination, most approaches focus on mimicking natural binocular depth perception using a stereo mount of two identical sensors. Due to their high temporal resolution and high dynamic range [15], event cameras, like the Dynamic Vision Sensor (DVS) [244] or the Asynchronous Time-Based Image Sensor (ATIS) [251], have great potential for depth reconstruction (see subsection 2.2.3). As stated in section 2.1.3, depth is inversely proportional to disparity δ, the difference in image position. This can be formalized as

$$\delta = X_L - X_R = \frac{bf}{Z}, \tag{3.1}$$

whereby depth Z refers to the vertical distance from the observed point. b is the distance from the center of the left camera to the center of the right one and f is the shared focal length of each camera. To obtain disparity, and therefore indirectly to regain the observable information about image depth, events of both sensors within a stereo mount must be assigned to each other. However, noise, faulty calibration, and possibly slightly deviating contrast sensitivity of the different sensors, lead to distortions and inaccuracies. Thus, despite the high temporal resolution, a purely temporal assignment of events would be almost utopian even with a perfect, ideal sensor. Hence, an exclusively time-based matching method, as visualized in Figure 3.1, produces a plethora of false positive matches.
Research in the field of computer vision, including depth reconstruction, has been frame-based for decades. As depicted in Figure 1.2, one can either transfer events to a frame-based representation, where the technical achievements of these sensors are largely lost or develop new algorithms that take their properties into account and thus profit from their advantages. This chapter addresses research question 1 defined in chapter 1: *"How can asynchronous event streams be optimally exploited for event-based stereo vision?"*.
A state-of-the-art for event-based 3D reconstruction is provided in section 3.1. Then, the core of this chapter is presented. A new approach for 3D reconstruction of event-based data using neural self-organization. For this purpose, the theoretical foundations of the approach are given in subsection 3.2.1 and 3.2.2. Subsequently, in subsection 3.2.3, 3.2.4 and 3.2.5, insights are given into additions

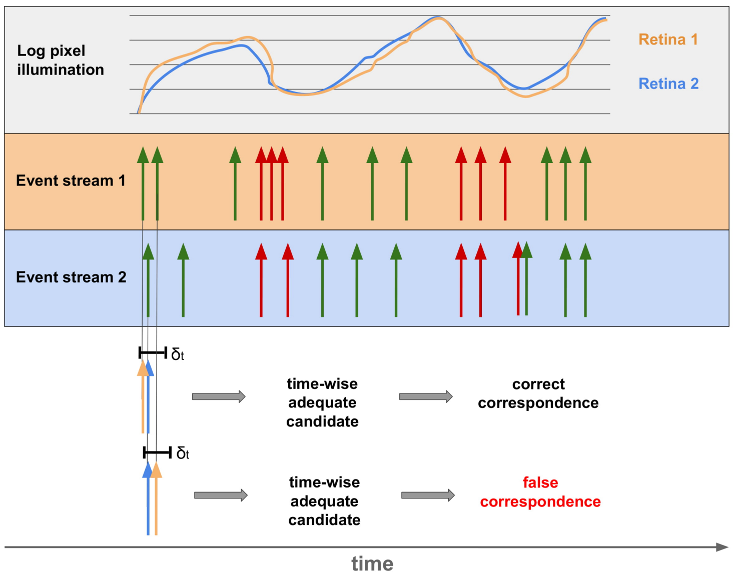


Figure 3.1.: Purely temporal matching of two event streams recording the same point in space. Exposure differences of corresponding pixels of both sensors are shown at the top. The second row shows the ON and OFF events for the first sensor and the third row respectively for the second sensor. At the bottom, it is illustrated that events within the time δ may be correct as well as false matches. Image source: (Steffen et al. 2019c)

to the basic approach that were necessary to improve speed and accuracy. This is accompanied by a qualitative evaluation of the basic approach without any extensions in subsection 3.3.1. In subsection 3.3.2, an introduction to data acquisition and sensor calibration is given. This is necessary to apply the method on an online demonstrator with real event cameras, as done in subsection 3.3.2. A comprehensive conclusion is given in section 3.4. Some of the material covered in this chapter was originally published by the author in (Steffen et al. 2019b). This concerns subsection 3.2.1, 3.2.2 and 3.3.1.

3.1. State-of-the-art

Even though event cameras have great potential for depth estimation, respective implementations are still relatively sparse [240]. One possible reason for this is that event cameras are a new field of research, while 3D reconstruction with frame-based cameras is decades ahead of the field. The most common form of event-based depth estimation is instantaneous stereo, whereby depth maps are created from short series of event streams recorded by multiple, but mostly two, rigidly mounted and synchronized sensors [15].

To transfer conventional approaches to event-based sensors, events can be converted to frames [260; 261; 262; 263]. As pre-processing consists of reconstructing gray values, disregarding the temporal correlation of events, the advantages of event cameras are lost in the process. Thus, these methods perform worse than algorithms processing events directly, due to a loss of temporal accuracy [264; 265]. Alternatively, researchers tried to apply handcrafted transformations from event sequences to frame-based representations, to enable the use of well-researched algorithms using deep learning and/or convolutional network types [266; 267; 268; 269; 270; 271]. Regarding the second option, to create new algorithms capable of processing events directly, first attempts comparing events by their timestamps and polarity [264] suffered from matching ambiguities. Another quite intuitive idea is to exploit epipolar geometry for 3D reconstruction [272; 273; 274; 258; 275], which is often also integrated into more sophisticated methods. Apart from the techniques of monocular and binocular vision, which are based on human vision, two other interesting methods implement completely different ideas. First, in (Schraml, Belbachir, and Bischof 2016; 276) stereoscopic panoramic imaging is proposed, whereby depth maps are generated using two rotating event cameras. Events are matched using temporal metrics in [276] and traditional stereo reconstruction is applied on intensity images in [276]. Second, an active approach using structured light is presented in [277; 250; 278]. These works differ in that all the works mentioned so far have passive methods of action, meaning, they do not interact with the scene. In contrast, [277; 250; 278] use a method in which light is emitted and the reflections are recorded by event cameras. In the following, first some monocular algorithms and then several stereo event-based stereo algorithms, mostly based on epipolar geometry, are introduced. Comprehensive surveys about event-based depth perception are provided in (Steffen et al. 2019c; 240).

3.1.1. Monocular Techniques

The first to tackle event-based depth estimation using only one camera is Rebecq et al. In [279], a method using Event-based Multi-View Stereo (EMVS) is applied, whereby the different viewpoints are created by moving the sensor, as visualized in Figure 3.2. The authors transfer the method for frame-based cameras using a space-sweep voting and maximization strategy [280] to event-based data. Thereby, exploiting the sparsity of the event stream without detours via event assignment or the generation of intensity images. The technique generates ray density volumes by back-projecting events into 3D. Structural properties of the recorded scene are given as the maxima of ray density. This idea seems very promising as event streams are well suited as input for the space-sweep algorithm due to highlighting edges by itself and triggering events from many closely spaced perspectives, far more than usually applied by a frame-based method. In EMVS the finalized 3D models are created by generating semi-dense depth maps from several viewpoints and merging them. In [281], a technique for monocular

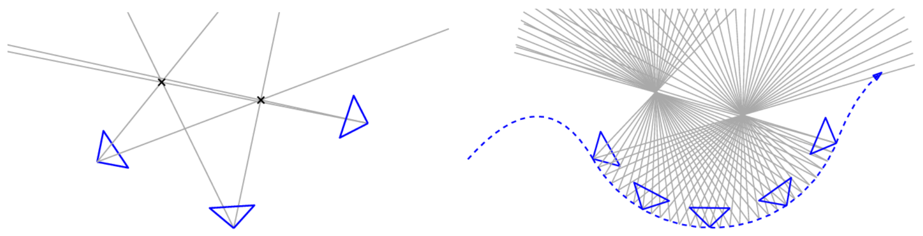


Figure 3.2.: Comparison of the conventional space-sweep approach using frame-based cameras at the left, with the event-based method at the right. In both images, two points are marked and rays represent back projection from these points. As frame-based sensors only provide a fixed number of views, only a small amount of rays is generated. The event-based sensor is moved along the dotted line and the areas of high ray density indicate the position of the points. Image source: [279]

3D reconstruction in real-time from solely a hand-held sensor is introduced. For this, three decoupled probabilistic filters are applied in parallel. To estimate the camera motion, a 3D map of the scene and an intensity image, only using event streams, are generated. The intensity image is hereby necessary to reconstruct depth. The generation of intensity images – grayscale values – only using event streams is unique. Integrated methods for grayscale reconstruction within event cameras usually rely on external data. Thus, ATIS uses per-event intensity measurement [251] and Dynamic and Active-pixel Vision Sensor (DAVIS) common intensity frames [250]. A simple but elegant framework to solve motion, depth and optical flow estimation using event cameras is proposed in [282]. Therefore, data association between the single events is regarded through maximizing the contrast of an image of warped events. Thereby events are warped regarding point trajectories which are described by motion parameters. This approach is the first in the field of event-based vision to tackle such a diverse range of vision tasks. In [283], a very different technique for event-based monocular depth estimation is introduced, whereby depth is obtained through focus and defocus. Using post-processing by a Spiking Neural Network (SNN), a depth map is calculated. It is assumed that an object with a uniformly changing focal length is initially out of focus. Thus, it has low exposure and becomes sharper over time, with the exposure increasing steadily. The applied event camera continuously triggers ON-events until the time of the best focus is reached and the exposure has reached its peak. In the further course, the focus and thus the exposure steadily decreases, which is why OFF-events are now triggered. This mechanism is visualized in Figure 3.3. The time of the best focus is determined as the average between two consecutive ON and OFF events. This calculation is done for each pixel individually. For this purpose, an SNN is created that contains five neurons per pixel. One neuron pair for triggering ON and OFF events and one neuron pair for suppressing the non-activated neuron of the first pair. For example, if an ON event is triggered, the matching suppressing neuron prevents the OFF event

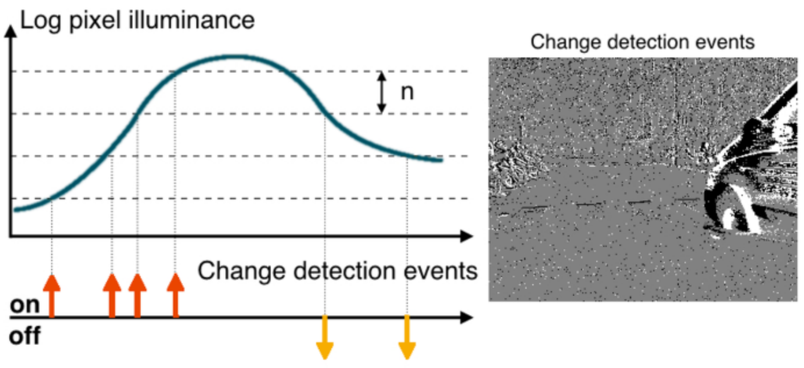


Figure 3.3.: The graph shows the change in exposure while focusing on a point. The point of best focus is directed at the high point of the graph. It can be seen that ON-events are triggered before this extreme point and OFF-events afterward. Image source: [283]

neuron from firing at the same time. Finally, an output neuron fires if two ON-events are followed by an OFF-event. The position of the object in space can be calculated by two timestamps. First when the focal length starts to change and second when the output is triggered. The authors claim that their algorithm is 20 times faster than real-time and very energy efficient. The Python package for neuronal networks (PyNN) framework and NEural Simulation Tool (NEST) are used for the simulation. A lens with motorized focal length control is needed for the setup. The author recommends a focusable liquid lens.

3.1.2. Stereo Techniques

For event-based stereo techniques, usually a hardware setup of two, or more, rigidly attached sensors is used. The sensor setup requires a shared clock, as temporal correlations can only be exploited if the events from both image planes are synchronized. Only then, the classical method of stereo reconstruction consisting of two parts is feasible. Firstly, epipolar geometry is applied to solve the correspondence problem and secondly, the location of the 3D point is triangulated [15]. However, the challenging and computationally expensive part is to find correspondences between events.

Cooperative Algorithms

Techniques inspired by human depth reconstruction are often based on the research regarding binocular vision by Poggio and Marr [124; 284; 285]. Their theoretical work was first implemented as cooperative algorithms with event-based sensors by Mahowald [286]. As these sensors do not create frames but event streams, they allow the exploitation of spatiotemporal information. Over a decade later, this work was continued independently by [52] and [53; 287]. The

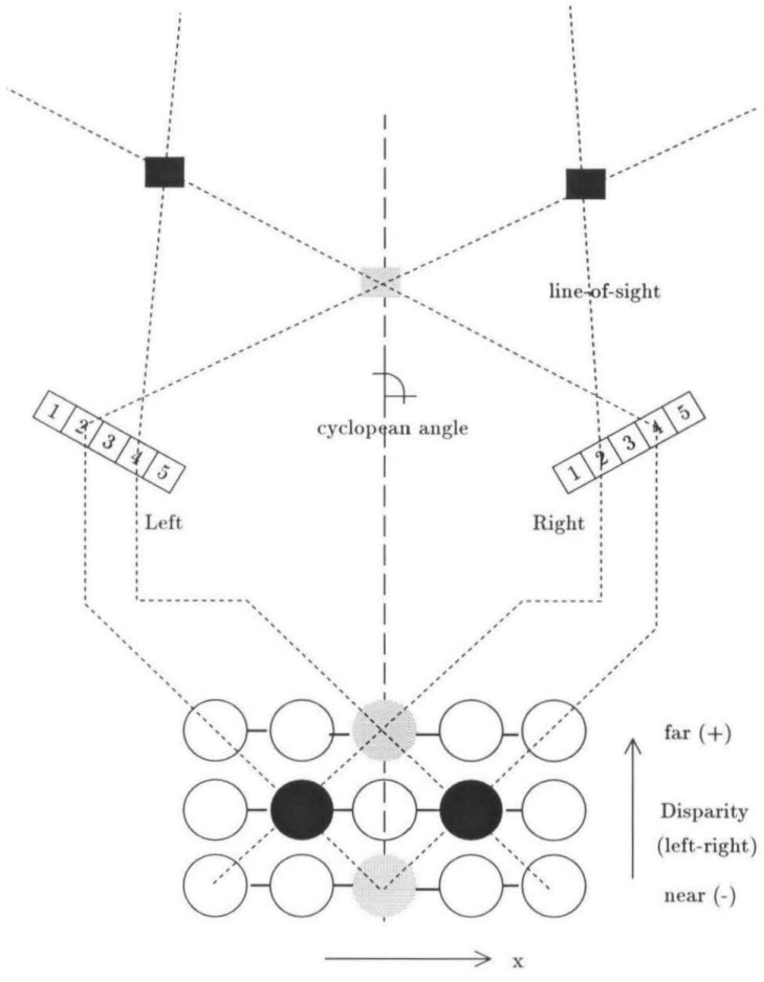


Figure 3.4.: Network architecture of a cooperative algorithm. Each retina is represented by five pixels observing a scene with two objects at 0 disparity. Additionally, a false target is shown at disparity -1. At the bottom, the circles represent correlator arrays, thereby, false matches are colored gray and matches black. Inhibitory interactions between these units, as defined by the matching constraints, are represented by dotted lines and stimulated once by solid lines. Image source: [286]

naming "cooperative algorithms" alludes to the fact that the neurons of the network, referred to as disparity sensitive neuron (DSN), communicate with each other and calculate disparity through cooperation. To add spatiotemporal relations to the temporal similarity the network dynamics of cooperative algorithms implement constraints that are derived from two assumptions postulated in [124]. *Uniqueness*, of each point in place at a given time and *Continuity*, thus that matter is continuous and divided into objects. For the technical implementation, DSN are fed with synchronized events from two sensors and extract disparity through inhibiting or exciting compounds, as shown in Figure 3.4, according to these matching constraints:

1. Inhibition between the communication of DSN in a vertical and horizontal direction, as each point from one sensor can only correspond to one point in the other sensor.
2. Stimulation of communication in a diagonal direction with the same disparity, as the matter is smooth and cohesive. Thus, the same disparity of neighboring neurons is amplified.
3. ON-and OFF-events inhibit each other if they occur at the same time in the same neighborhood because changes in lighting are less common for spatially and temporally close events.

These principles are implemented with spiking neurons, which process event-based data ideally, in [54; 55]. Three components build up the network in [54], OFF- and ON-neurons (retina coordinates), coincidence detectors and disparity detectors. Due to their cell arrangement, by emitting spikes, coincidence detectors encode positions of a possible disparity caused by simultaneous spikes of retina coordinates. The matching candidates gathered by coincidence detectors are subsequently evaluated by disparity detectors which implement C2 and C3 by respective stimulation and inhibition and C1, by the disparity detectors' recurrent, inhibiting synapses. Hence, retina coordinates encode changes in lighting, coincidence detectors detect temporal correlation and disparity detectors implement biologically plausible matching constraints. In [54], it is reported that high-frequent stimuli generate false correspondences, due to DSN simply accumulating signals of both retina coordinates and spiking if a threshold is exceeded. Micro-ensembles, realizing a logical 'AND' are introduced in [55] which suppress spikes if the input is unbalanced, therefore only from one side. Due to the principle of action of event cameras, in [54; 55], disparities is merely detected from dynamic scenes. This is extended to static scenes in (Kaiser et al. 2018) by applying microsaccades, small high-speed panning or tilting motions of the stereo setup.
Cooperative algorithms were extended using Gabor filters [288; 289] and belief propagation [290; 291; 292; 293]. In [288; 289] Gabor filters are applied to the raw event streams before passing them as input to the cooperative network, allowing exploitation of matching edge orientations of sensors as a supplementary criterion. The methods introduced in [290; 291; 292; 293] using belief propagation, transform the correspondence problem into a labeling issue, whereby labels represent disparities.

Generalized Time-Based Stereo

A spike-based method that uses a four-step algorithm to compute the probability for two events to be triggered by the same 3D point is introduced in [258]. As visualized by the flowchart in Figure 3.5(a), the algorithm uses criteria such as energy functionals. The final probability is determined as the minimized sum of the energy functionals. Before the energy functionals are calculated, event pairs must first surpass two thresholds; Each possible match of events must be no further apart in time than ϵ_T and cannot deviate more from the geometric constraints than ϵ_S pixels. For the *(1) temporal criteria* and the *(2) spatial criteria*, the energy

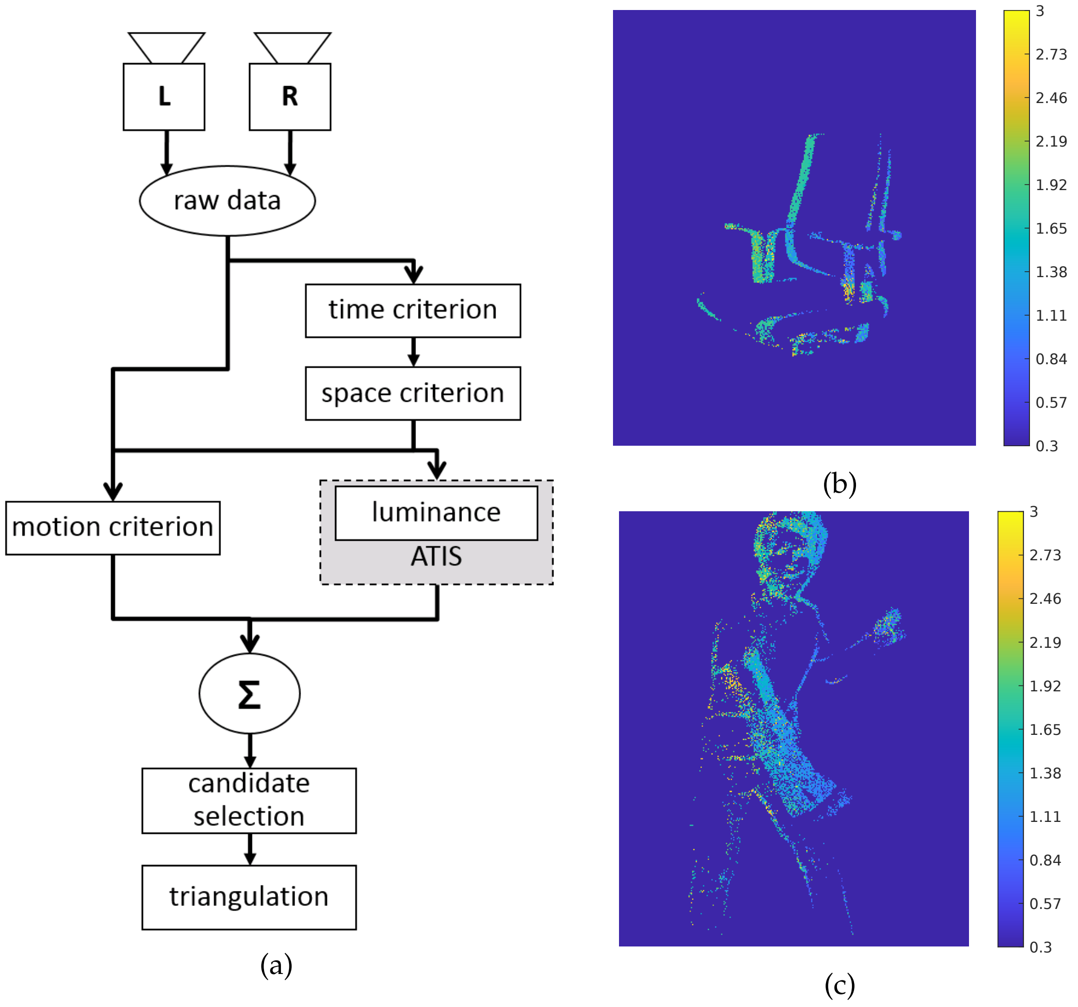


Figure 3.5.: Flowchart and depth maps of the method "Generalized Time-Based Stereo". (a) Four criteria are used in [258] to establish correspondences: *(1)* events' timestamp, *(2)* epipolar geometrical properties, *(3)* luminance values and *(4)* motion fields. (b) & (c) Corresponding depth charts, distances are represented by colors as stated by the color bar. Image source: (Steffen et al. 2021a; Elfgen 2020)

functional is 0 for perfect matches and 1 for differences that are within tolerance. The first criterion simply compares timestamps, while the second one makes use of techniques presented in previous algorithms [272; 274; 273] exploiting epipolar geometry. For the *(3) motion criteria*, a motion surface of previous adjacent events is generated for each of the possible matches. The corresponding energy functional is lower the more similar these motion surfaces are. Lastly, the *(4)* *luminance criteria* compares the respective changes in gray values that are associated with both events. This criterion is only applicable to an ATIS, as it utilizes its asynchronous mechanism to generate gray values. The ATIS sensor, as visualized in Figure 2.20 has two circuits for each pixel, one generating a transient response in the form of an event stream (Change Detection (CD)) and one obtaining a sustainable response in form of grayscale images (Exposure Measurement (EM)). This algorithm [258] was implemented and evaluated in the course of this thesis. Its application on a stereo setup of two ATIS generated the depth maps shown in Figure 3.5(b) & (c).

Implementations on Dedicated Hardware Solutions

Mots presented algorithms make use of standard processors, more precisely CPUs. This applies to early methods as [260], which uses a simple frame-based algorithm to generate disparity maps, and [273], solving the correspondence problem with two time-windows implemented with the "Java tools for AER (jAER)" software. However, Central Processing Unit (CPU)s are also common among more biologically plausible and advanced methods as cooperative algorithms [53; 287; 52; 289; 291] and spiking techniques as [283; 290; 54]. However, some rare implementations for neuromorphic processors [55; 294], Graphics Processing Unit (GPU) [295] and Field Programmable Gate Array (FPGA) [296] exist. As event streams, generated by event cameras and sent by Address Event Representation (AER), are well suited as direct input for an FPGA, an integrated setup does not bring any additional latency thus maintaining the extremely high temporal resolution [240]. Hence, this combination might become common shortly. However, an implementation for event-based depth reconstruction on FPGA, as presented in [296], is so far still very rare. The proposed method uses a stereo setup of two ATIS and solves the correspondence problem for image blocks of 1616 pixels instead of individual pixels. In [295], an approach, based on [279; 282], for space-sweep on GPU is introduced. This work applies sensor motion for synchronizing the events of both event cameras, to create an event disparity volume which represents correct disparities in focus and false once blurred. This method is a brute-force variant of space-sweep with a new technique for a relatively low matching cost. Implementation of cooperative algorithms, as introduced in section 3.1.2, using a neuromorphic chip is presented in [55] for spiking neural network architecture (SpiNNaker) and in [294] for TrueNorth. The system in [55] estimates depth with a latency of 2 ms but its power consumption of 90 W makes the SpiNNaker implementation not applicable for real-world scenarios [240]. In [294] nine TrueNorth chips are applied and the low-power system achieves an average of 400 disparity maps per second.

Related Frame-based Methods

In stereo vision, 4D data (2x2D) is mapped to a 3D representation. As the Self-Organizing Map (SOM) is often used for dimension reduction, it can be applied to solve the correspondence problem. This idea was first implemented on frame-based data in the '90s by [189; 190]. The method proposed in [190] is based on the theory of Marr and Poggio [285] that true correspondences satisfy three constraints, derived from properties of physical objects: similarity, smoothness and uniqueness. Hence, this work is related in its theory with techniques presented in section 3.1.2. However, in [190] the correspondence problem is solved by a learning strategy based on the SOM algorithm. Thereby edge segments are used as features and similarity constraints, determined by the minimum squared Mahalanobis distance, for establishing correspondences. In [189] a quite different approach is taken, depth maps are generated as the current state of a dynamic process. Possible disparity values are attributed to every point, subsequently the correct disparities are selected with competitive learning via a SOM. The authors claim that their idea has strong analogies to the human visual system and can be implemented with massive parallelization. For stereo vision on frame-based data Scale Invariant Feature Transform (SIFT) is a popular method that is incorporated in many respective algorithms. In [297] the authors propose to apply a SOM to perform the matching of SIFT more efficiently. The result is lower calculation times and a doubling of the matched features. Stereo vision through self-organization was also used for estimating 3D hand poses [191] and generating 3D face models [192], both from stereo images. Furthermore, learning-based approaches are successfully used in frame-based depth estimation, especially applying Convolutional Neural Network (CNN) [298; 299; 300]. A very potent deep learning method for stereo vision was presented in [301]. The authors claim that this method only requires very few images to reconstruct the geometry of an object either as a depth map or as an occupancy grid. The architecture is built on CNN and internally represents the environment as a discretized 3D grid. Thereby differentiable operations are used for projecting and unprojecting, which enables end-to-end learning. In this way, the underlying 3D geometry is learned in a metrically accurate manner.

3.1.3. Discussion

In recent years, many different methods for event-based depth perception have been presented. A fair comparison is difficult because these techniques are not evaluated on a uniform data set. Such a data set would be very difficult to create as the different methods – monocular, stereo, structured light, and multi-perspective panoramas – differ greatly from each other in terms of data acquisition. However, one conclusion is uncontroversial; for event-based data conventional algorithms, created for frame-based cameras, perform significantly worse than time-based algorithms [264]. Although interesting, stereoscopic panoramic

imaging and methods applying structured light are not well suited for this use case. The former is well applicable if a 360° view is needed, but less suitable for reconstructing a defined work cell. The latter, active techniques, show problems with different reflection properties of materials, which is critical when used with robots. Regarding monocular techniques, the method of Rebecq et al. [279] and Kim et al. [281] are quite potent, and especially [279] also quite efficient, but the setup is not ideal for the use case intended here. With EMVS and also the handheld camera in [281], a static scene is captured by a constantly moving sensor. However, the work presented here attempts to capture motion and changes in a defined workspace through a fixed camera mount. Also, the techniques presented in [283] and [282] are interesting, however, monocular depth estimation suffers from occlusion much more than stereo vision, as two viewing angles capture the space better. Additionally, depth by defocus [283] requires additional hardware. In general, monocular methods solve a fundamentally different problem than stereo techniques, as one cannot leverage temporal correlation between events of multiple images. The methods [279; 281; 282] create a 3D edge map, hence, a semi-dense 3D scene reconstruction. To obtain information from events of one moving camera, these methods require knowledge of camera motion and generally more time, a longer interval, than stereo methods [15]. Due to this, they are often applied for Simultaneous Localization and Mapping (SLAM) [302; 303; 304; 282] but do not perform well for use cases requiring instantaneous depth estimation.
A comparison of different stereo depth estimation techniques, regarding their characteristics and performance, is provided in [294] even if very new methods are not considered here. The technique introduced in [258] looked extremely promising. For one, it systematically builds on many previous algorithms [272; 274; 273] that make good use of epipolar geometry. Also, besides making use of a technical realization of magnocellular pathways, using event streams, the approach in [258] uses a technical implementation of parvocellular pathways through using the ATIS' EM circuit for obtaining gray-scale images. The luminance criteria which is based on EM restricts the algorithm so that it can only be used on a stereo setup of ATIS. EM to generate gray values has not been implemented for DVS or DAVIS, and to the author's knowledge, no other sensor. However, the authors of [258] state that this fourth criterion has very little impact on the overall performance. As the algorithm was implemented and tested (see Figure 3.5(b) & (c)) during this thesis, it could be determined that it is not optimally suited for this use case. The performance was satisfactory, but, the accuracy was not sufficient to represent a robot's working cell for subsequent path planning. As event streams are processed efficiently and in a way that preserves data well by SNN, and the strengths of SNN can only be fully exploited on dedicated hardware, solutions presented in section 3.1.2 were of particular interest. The advantages of this pipeline are shown in [296], a method solving stereo vision with an event camera and FPGA. It minimizes the computational load as well as the necessary memory access [240]. Although neuromorphic chips are currently making extreme strides in development, and are now also supported by the industry, prototypes were used in the methods presented. Its implementation on

neuromorphic hardware is very promising but also brings great challenges. That is why there are only a few such implementations as of today. These are mostly very specialized for difficult-to-access one-purpose hardware. Furthermore, in terms of performance, these techniques are still far away from what is achieved by conventional methods. This can and very likely will change significantly in the coming years.
All work presented in section 3.1.2 is based on frame-based image acquisition. However, both, the learning-based approach of [301] and the general idea to see the correspondence problem as a dimension reduction issue that can be solved using SOM, are very promising. As known for stochastic algorithms, the SOM is especially well suited for data streams [91], which makes it a great fit for event cameras. Furthermore, the ability of [301] to create a voxel occupancy grid, is very interesting for the research goal discussed in section 1.2. However, since the work of [301] is based on CNN, its network architecture embodies many layers. Also, the method requires relatively large quantities of training samples. To the best of the author's knowledge, there is no approach using a SOM which is similar to the work presented here. In particular, no methods are mentioned in literature applying a SOM for event-based data streams to solve the correspondence problem.

3.2. 3D Reconstruction through Self-organization

Most potent techniques for frame-based stereo vision apply a learning-based representation. However, the majority of event-based methods in section 3.1, which are not transferred from a traditional frame-based approach, either use grid-based models or a simple form of event representation [305; 268]. Although it is of course not feasible to transfer a conventional learning approach, it is promising to develop a suitable learning approach for neuromorphic data. A huge disadvantage of stereo vision is the need for calibration, as shown in subsection 3.3.2 in the paragraph regarding data acquisition and calibration. Most techniques require precise calibration of intrinsic as well as extrinsic parameters [306], thus creating a source of error [240], and negatively and often strongly affecting the quality of the reconstruction. Many of the methods presented in subsection 3.3.2 regarding calibration are quite complex as they implement multiple complementary constraints, such as [258; 54; 55]. Thus, the introduced systems are difficult to reconstruct and maintain, and often also require very specific hardware. In contrast to related works, this research focuses on a learning-based method for event-based stereo representation without the requirement of before-hand calibration.
The system is trained with random data points from within a defined 3D space. After training, the SOM has adapted to represent the entire 3D space evenly. The individual neurons of the SOM are sensitized during training to simultaneous activation of pixels or even areas on the sensor. Thereby, the neurons of the SOM

learn which pixels in the left and right sensors belong to which voxel, representing a cubic subspace in the defined workspace.

3.2.1. Biological Derivation and Rationale

Even though, artificial stereo vision is very successfully applied, in terms of energy consumption and latency it is far inferior to the biological model. Therefore, a bio-inspired argument explaining 3D reconstruction in the cortex is used here to tackle artificial stereo vision. Using fMRI data of macaque monkeys two cell types in the ventral premotor cortex (F5) could be related to the viewing and grasping objects, canonical neurons and mirror neurons [307]. First, canonical neurons are activated when looking as well as grasping an object, thus the same brain areas are responsible for observing a specific object as for executing movements in response to this object. However, if no grasping is performed, the canonical neurons will only fire for objects within the peripersonal space [308]. Second, mirror neurons fire equally when the monkey itself grabs an object and when it watches another monkey doing so. The better known mirror neurons refers to the neuronal execution of movement, by watching, a new movement can be learned. In contrast, canonical neurons are activated while observing an action-related object in reach. They are involved in the recognition of an object's 3D shape but do not play a role in object identification. Further insights, also regarding the extent to which this can be transferred to humans, are given in [307]. Another aspect of this approach modeled on nature is the application to event-based data (see subsection 2.2.3). Thereby, a new physical constraint – time – is exploitable, to solve the matching problem in stereo vision. Furthermore, a biologically plausible, yet simple, self-organizing network structure (see subsection 2.1.2 & 2.2.2) is implemented through a SOM. In general, Self-organizing Neural Network (SONN) are inspired by neural self-organizing structures, so-called feature maps which are located in human and animal cortices [96]. In section 3.1.2 it was shown that self-organization was used to solve the correspondence problem on frame-based data already in the '90s.

As the SOM functions as a dimension reduction, it can be applied to stereo imaging, consisting of 4D (2x2D) or higher, to recover the underlying latent 3D representation corresponding with the real-world spatial coordinates. The SOM's weights impose a topology that can be applied to learn a mapping to the Cartesian space. The dimensionality of the input data depends on the number of cameras used as image coordinates are concatenated. Formalized, the dimension is $2 \cdot k$ cameras with $k > 2$. Thus, the input space is 4D for stereo and 6D in case a third camera is used. How the proposed system implements these biological aspects in its architecture, thus realizing a structure with analogies to the human visual processing system, is visualized in Figure 3.6. Thereby, the input data is rasterized into pixels and the network uses this division as one neuron represents a single pixel or a group of adjacent pixels. Each node of the 3D grid, the SOM's neurons, connects to one or several of the retina neurons, representing the pixels

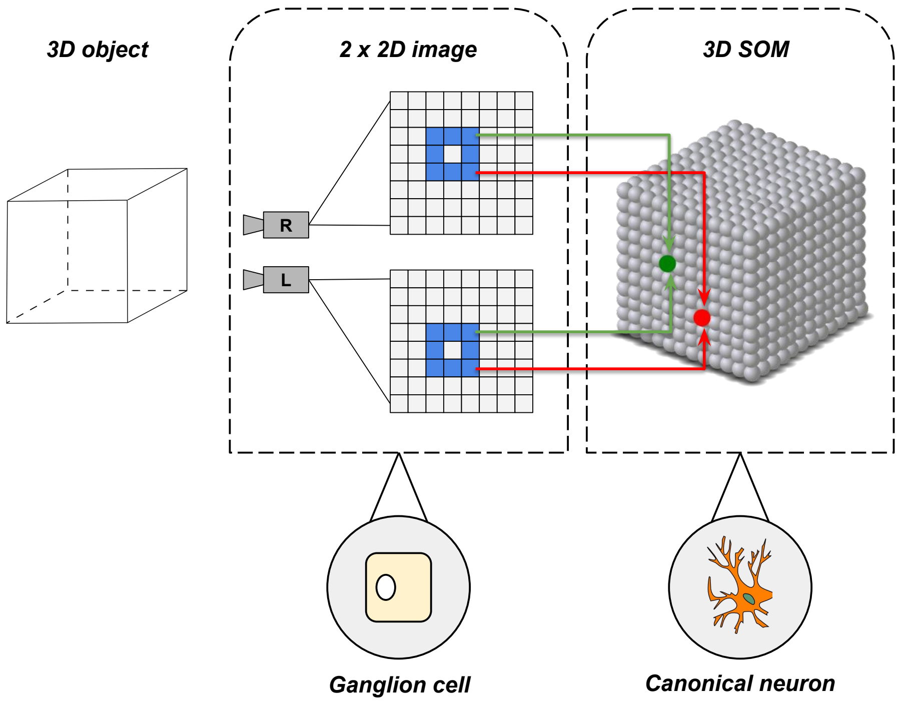


Figure 3.6.: Concept sketch of the biological motivation to use self-organization for solving the correspondence problem. A cube is recorded from two slightly shifted perspectives. The corresponding points of both 2D images are determined by the use of a 3D SOM, which maps the 4D image data to a 3D representation. The inspiration, therefore, comes from the canonical neurons populations and their object depiction. Image source: (Steffen et al. 2019b)

of each sensor. These connections are encoded in the weights of the traditional SOM pointing to pixel coordinates. Respective retina neurons are activated if the pixel, or group of pixels, they encode is active. A 3D cube is perceived as slightly shifted 2D squares by two sensors, which mimics ganglion cells, as introduced in section 2.1.3. The method learns how to reconstruct a 3D representation from 2D data recorded from multiple viewpoints by using a Kohonen SOM, which subsequently can be applied for correspondence detection. Thereby, the learned SOM functions, in an abstract way without precise biological conversion, similar to how canonical neurons are assumed to behave, by obtaining an object's 3D space occupancy. The SOM is a multidimensional grid structure in which each vertex is a neuron embodying a weight vector in the high dimensional feature space [178]. For each learning sample, a Best Matching Unit (BMU) is chosen, and the weight vector that is closest to the sample. Hereby, the most similar weight vectors, and, with less intensity, also their neighbors, are shifted in the direction of the input. To implement a transition from exploration to exploitation, the BMU's sphere of influence is continuously reduced during the learning process. This is realized in the basic SOM algorithm, as discussed in section 2.2.2. Two parameters can be

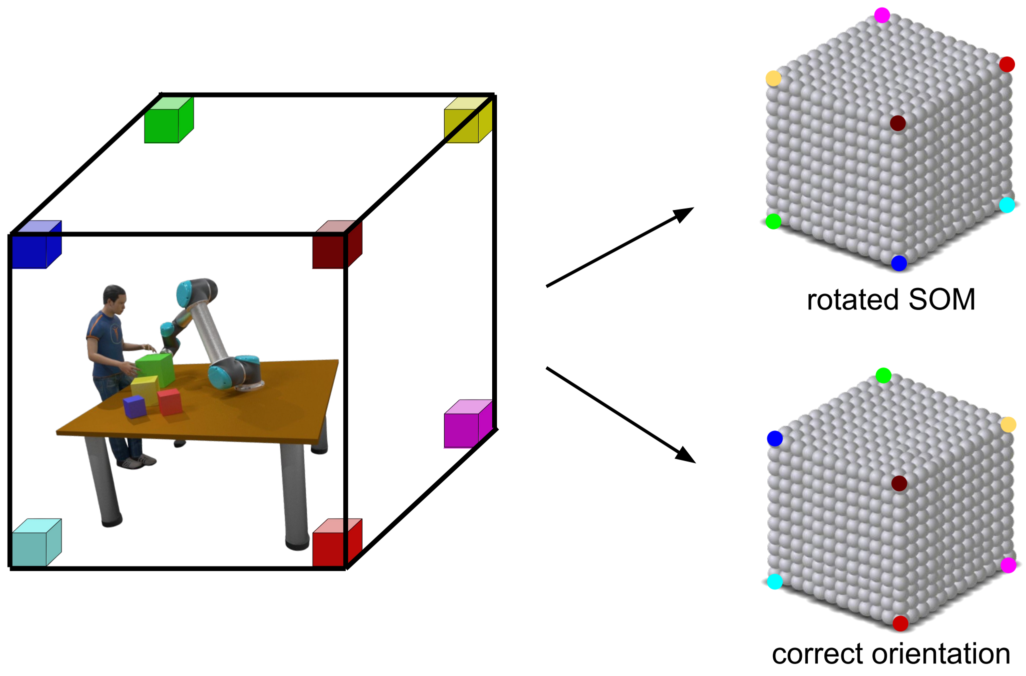


Figure 3.7.: A learned SOM samples the workspace, which it represents. The neurons are organized in a 3D grid and thereby a voxel space is created. Each neuron corresponds to one tiny cube in the 3D space, as visualized here through colors. To map the Cartesian space the topology of the SOM's weights is used. However, the orientation of the trained SOM is not stable, it might change after each training run. The disoriented SOM shown is only an example, theoretically any form of mirroring and rotation can be present. But, as can be seen here, the neighborhood relations remain intact.

adjusted to manipulate this effect, the learning rate η and the neighborhood rate σ, see Equation 2.9 and 2.11 respectively.
The proposed approach interprets the stereo correspondence problem as a dimension reduction, whereby the SOM induces a topological structure and parametrization from unlabeled data sampled from a latent manifold. The neurons within the network each correspond to an area within an abstract 3D occupancy grid. Thereby, each neuron is sensitive to a specific combination of camera coordinates and represents a segment of the 3D space. In Figure 3.7, the workspace to be sampled is seen in the form of a cube. Each voxel in the workspace volume corresponds to a neuron of the trained SOM. The coordinates of the grid can be translated into 3D coordinates of the real world. That works since SOMs are well suited to find properties with the same dimensionality as the SOM but embedded in a higher dimension. However, it is not possible to influence how the neurons of the SOM map the manifold, that is, which neuron corresponds to which point in 3D space. Thus, the orientation of the SOM is not stable, as formally derived in subsection 3.2.2, meaning it can be rotated and mirrored. If the SOM is trained,

there are eight neurons which could be each corner of the 3D space. Yet, this does not pose a problem for most applications, for example regarding collision checking the orientation is irrelevant. This is only a major problem concerning visualization and evaluation. Thus, relevant methods are not directly affected, but their experimental validation is more difficult. The neurons' neighborhood relations represent connections on the manifold. Therefore, if no topological mismatch is created, see Table 2.2 & Figure 2.14, the manifold's parameters are represented by the neurons' indices. The underlying manifold parameterization, which is created through feature mapping by self-organization, makes this technique agnostic to calibration parameters. Hence, extrinsic and intrinsic camera calibration as well as lens distortion are implicitly learned and therefore encoded in the final mapping. This method, as it is quite computationally expensive, achieves good performance through massive parallelization. However, this has to be considered in relation as it is not a gigantic Artificial Neural Network (ANN) with many parameters and billions of neurons. The memory requirements are $\mathcal{O}(n3^)$. Finally, that most involved operations can be easily parallelized is also the prerequisite for further acceleration through dedicated technologies such as neuromorphic hardware or GPUs.

3.2.2. Formalism of Topology Induction

The formalization of the approach is to be done by employing an example. If random samples are generated from the surface of a sphere and used as input for a 2D SOM, the SOM's weight vectors are uniformly distributed over the surface of the sphere after learning. This can be interpreted as the surface being unwrapped by the neural map and parametrized by the indices of its neurons. The definition of the sphere, more generally the topological space or 2D manifold, can be done by a set of points:

$$M = \{(x, y, z) : f\left((x, y, z)\right) = 0\} \tag{3.2}$$

given the implicit form $f(\cdot)$ of the manifold. In case the SOM converges, it maps an ordered pair, referred to as indices, to points on the manifold:

$$(u, v) \mapsto \omega_{u,v}, \ \text{ such that } f(\omega_{u,v}) \approx 0 \tag{3.3}$$

and

$$g(\omega_{u,v}, \omega_{a,b}) \leq g(\omega_{u,v}, \omega_{c,d}) \Leftrightarrow \|(u, v) - (a, b)\| \leq \|(u, v) - (c, d)\| \tag{3.4}$$

thereby each pair of points on the surface is connected by $g(\cdot, \cdot)$. The discrete parameterization of the manifold, or more specifically sphere, is consequently given by the coordinates (u, v) and the vectors $\vec{u}, \vec{v}$. Even though the algorithm does not necessarily converge, in almost all cases it does eventually converge on a set of

neurons, reflecting the underlying distribution of the input vectors stem (Tatoian and Hamel 2018; 309). In this context, convergence is seen as a stability criterion. If a SOM converges, this means that after a certain number of training runs, new samples are inserted at the same position, even if training is continued. A proof of convergence for the majority of the neighborhood functions is also provided in [196]. It is noteworthy that the orientation of the map can change during each training run, which does not violate the convergence as long as the cluster arrangement is stable. However, that means that SOMs by their nature are non-deterministic in the sense that the directions $\vec{u}$ and $\vec{v}$ may swap as well as the signs may change. The impact of this on the present use case is shown in Figure 3.7, including the resulting consequences for this approach. This is true for the original SOM version by Kohonen, which is stochastic, but, the Batch Self-organizing Map (Batch SOM) and its derivatives are deterministic [197], as stated in Table 2.2.
The dimension of the example presented above must be increased for the approach to 3D reconstruction. As stated in subsection 3.2.1, the number of cameras k determines the dimension of the input space which is $2 \cdot k$, thus 4D in the case of stereo vision. The latent manifold which is to be recovered is the Cartesian space:

$$M_{\text{Cartesian}} = \left\{(x_1, y_1, \ldots x_k, y_k) \in \mathbb{R}^{2k} : f(x_1, y_1, \ldots, x_k, y_k) = 0\right\} \tag{3.5}$$

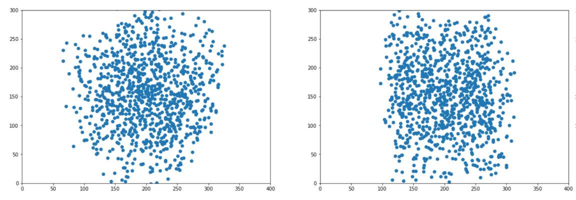
If the pair of image points in question corresponds to the same point in the real world, the following equation for the indicator function is true: $f(\cdot) = 0$. The properties of the 2D case stated in Equation 3.3 & 3.4 become for 3D:

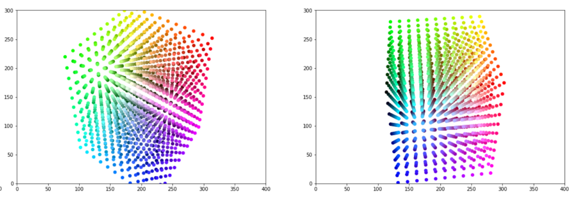
$$(u, v, w) \mapsto \omega_{u,v,w}, \quad \text{such that } f(\omega_{u,v,w}) \approx 0 \tag{3.6}$$

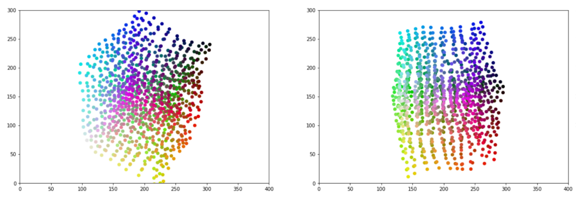
and

$$g(\omega_{u,v,w}, \omega_{a,b,c}) \leq g(\omega_{u,v,w}, \omega_{d,e,f}) \Leftrightarrow \|(u,v,w)-(a,b,c)\| \leq \|(u,v,w)-(d,e,f) \tag{3.7}$$

Using simulated data, an example is given in Figure 3.8, to illustrate the method vividly. 3D points of a cube and their corresponding image points are used as training input. The lattice of the SOM is also a cube with an edge length of 10, embodying 1000 neurons. The graphic shows how after learning, when the SOM has converged, its weights represent the Cartesian space and a parametrization is induced by the neural weights. In Figure 3.8(a) the first 1K samples of the input space are visualized. The input data are randomized samples of a $1\,\text{m}^3$ cube. The target representation of the weight vectors, thus the ground truth for evaluation purposes, is visualized in Figure 3.8(b) and the actual weight vectors are shown in (c). To make the exact contours of the neuronal lattice visible, only the edges were visualized in Figure 3.8(d). The color coding reveals that the axes have not remained stable but have swapped, exactly as expected. However, it is not complicated to determine the correct transformation in case orientation is needed, as done in subsection 3.3.1. At this point, it should again be emphasized that no calibration is necessary for the 3D reconstruction. Calibration is only used


(a) samples


(b) ground truth


(c) weight vetcors

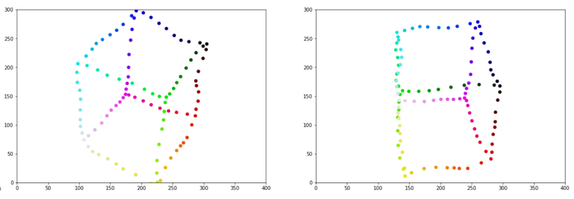

(d) Edges of neural hypercube

Figure 3.8.: The weights of a converged SOM represent an evenly sampled Cartesian space and the neural coordinates induce a parameterization. (a) The first 1K samples (from a total of 10k) of the input space represent a cube from two slightly shifted perspectives. (b) Target representation showing a color coded 3D lattice. (c), (d) The actual weights in the SOM's feature space. Image source: (Steffen et al. 2019b)

to obtain the camera model in simulation and as ground truth. The number of neurons grows in $\mathcal{O}(n^3)$ and is crucial as the resolution and therefore the accuracy of the results depends directly on. In the case of a workspace with $1m^3$, as could be used for the scenario in Figure 1.3 of section 1.2, it is sufficient if the neurons cover a voxel with edge length $5cm$ of the 3D space.

3.2.3. Detachment of Learning by a Prelearned SOM

A SOM embodies two processing steps, or modes, training and matching. One way to significantly speed up the algorithm on a real-life application is to use a network that was pre-learned in simulation, with a continuously decreasing learning rate. Hence a learning phase in simulation is very beneficial before applying the algorithm to real hardware. For this purpose, the size of the actual workspace of the intended use case is required. For training the SOM, random samples of a cubic space serve as input. The cube has to be well visible from the perspective of both sensors and its size must correspond to the targeted workspace. To do this, in practice, the work cell and stereo setup is replicated in simulation using a Unified Robot Description Format (URDF), as shown in Figure 3.15(a). Thus, the work cell's size and pose in respect to the sensors can be extracted from the URDF. The samples are projected into camera coordinates of the left and right sensor by use of the calibration matrix, which embodies the camera parameters of each sensor individually and also a stereo calibration. The parameters encode the orientation and position of the sensor setup in space. Consequently, changes in orientation or position can make the prelearned results ob-

solete. This is especially true if the cameras are shifted in relation to each other but may also occur if the workspace changes in size or, even worse, shape strongly. Prelearning has to be repeated with the correct calibration matrix for the new setup. However, including the camera setup into the system allows for rapid pre-learning with a conventional SOM. This alleviates the difficulties often encountered with setting up event cameras and removes the need to implement more complex training solutions involving SNN, which reduces the time needed to get the system running.
The neurons' weights represent the image coordinates for both cameras, thus all neurons hold weight vectors $w_i = (x_1, y_1, x_2, y_2)$. These weights are trained to represent the specific reactivity to camera events for individual neurons which makes the SOM approximate the sampled space, in short, it "matches" the cube. In Figure 3.9 a training succession is shown explanatory for a $6 \times 6 \times 6$ SOM and

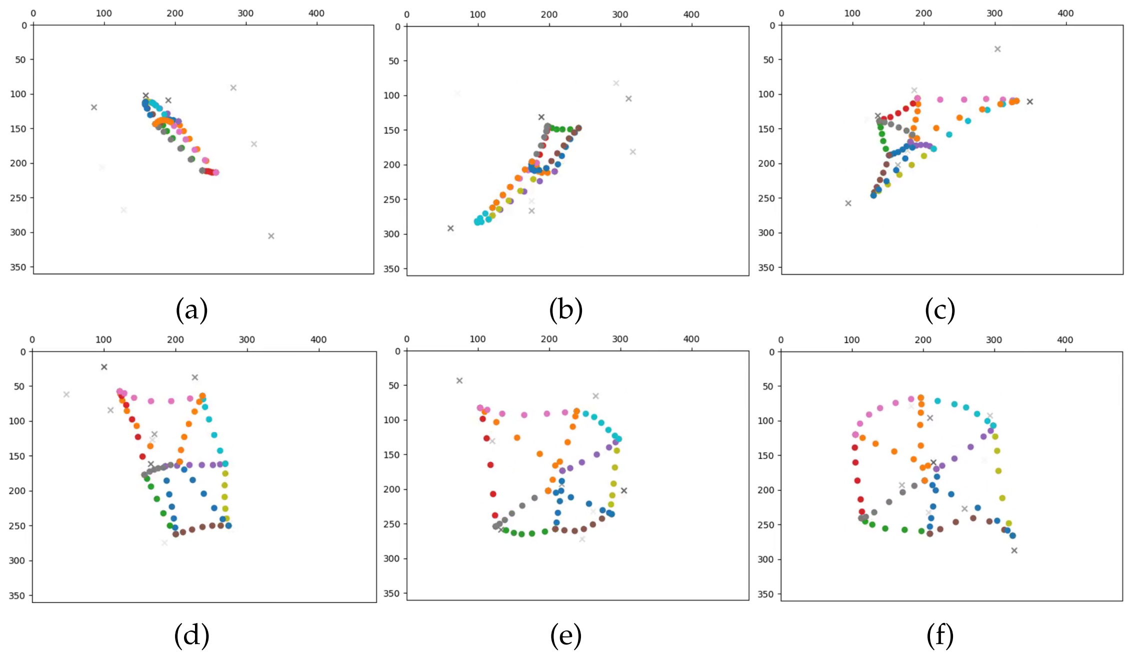


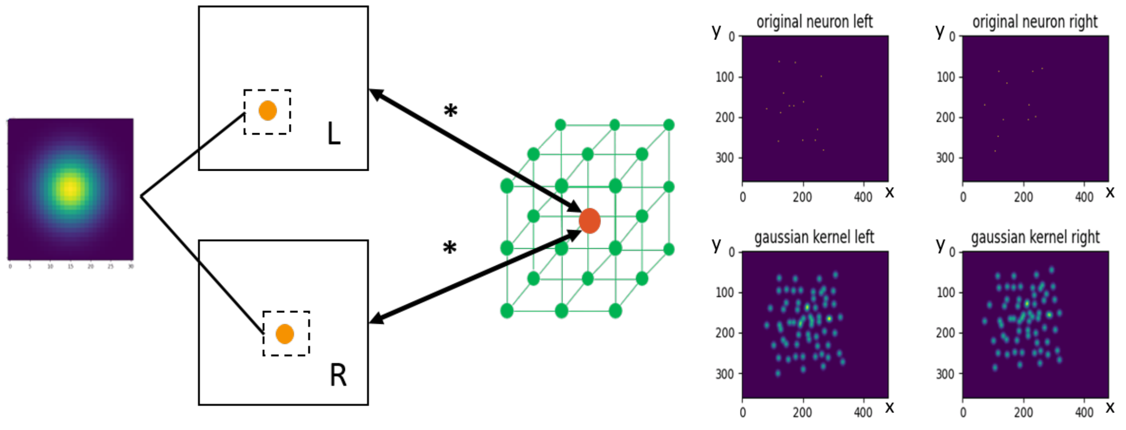
Figure 3.9.: Development of a SOM during prelearning. The gray x are the input samples and the development of the SOM is shown in color. The image series was taken with the interval of 4 seconds and (a) is the very start of the training phase.

correspondingly small workspace. Like for Figure 3.8 two perspectives of the cube are recorded. However, the visualization is limited to the left one because a second viewpoint does not provide any additional information here.
It would not be necessary to retrain a SOM which was pre-learned on simulated data drawn from a cube on the actual application, thus the applied stereo setup and the real-life work cell. However, this is strongly dependent on the quality of the calibration, which was obtained from the actual camera setup and applied to the simulated camera model. As calibration for event-based sensors is not trivial, it is safe to assume that retraining is very beneficial, as shown in the paragraphs

regarding calibration in subsection 3.3.2. However, training on the actual application, with real cameras and a real scene is far from trivial. To preserve the shape of the workspace, the cube representing the robot's work cell would have to be sampled evenly from this area. The lattice of the SOM will otherwise adapt to the environment. Therefore, if the cameras detect movement outside the workspace, an adjustment would take place for this and the SOM would change its shape. Furthermore, in the case of normal distributed data points for the training phase, the SOM will shrink, as everything will be drawn closer to the center.

3.2.4. Receptive Fields

The method proposed in subsection 3.2.2, which is evaluated in subsection 3.3.1, uses all neurons as possible matching candidates. Hence, matching is performed all-to-all and the BMU is selected from all neurons of the network, which harms the performance. To narrow down possible candidates synaptic fields are gener-



(a) (b)

Figure 3.10.: Gaussian kernels acting as receptive fields for increased matching speed. (a) A sketch to illustrate the concept. (b) Exemplary representation in the retina space. The original neurons are visualized at the top and their associated receptive fields at the bottom, for the left and right viewpoints. As a SOM with edge length 20 was used and the data fit an ATIS with resolution 480×360, the resulting data format is $20 \times 20 \times 20 \times 480 \times 360$.

ated for each neuron. By applying a function for back transformation, weights for the connection of each neuron in the SOM to each pixel in each camera can be determined. Instead of a 4D element, like the input weights of form $w_i = (x_1, y_1, x_2, y_2)$, hereby arises a 5D data format. The 3D coordinate of the SOM's neural grid is supplemented by each axis of the image plane. In the case of the ATIS, the resulting data format is $X \times X \times X \times 480 \times 360$, whereby X is the edge length of the SOM. This function can generate a one-to-one mapping between pixels and neurons. However, by applying a Gaussian "filter" around the pixel with the strongest influence towards each neuron, a more sophisticated connection matrix is applied. These connection matrices called *"receptive fields"* from now on, connect the pixels of both event cameras to neurons of the SOM in a natural fashion. Thereby, every neuron could be connected to all pixels, albeit most connections have a weight vector of 0. However, this depends heavily on the chosen size of the receptive fields. In case a reasonable size is chosen for the Gaussian kernel, as visualized in Figure 3.10(a), neurons are matched only locally. Furthermore, matching with synaptic fields generalizes the approach. For one, it becomes more suitable for application to SNN, as it is more in line with their sparse concepts. Also, it allows the processing of frame-based data for which comparing each pixel of one image with the other would be necessary without this method. The receptive fields are placed over the field of view, similar to a CNN, as shown in Figure 3.10(b).

3.2.5. Bootstrapping by Shape Segmentation

The concept of bootstrapping is inspired by human ontogenesis [310]. It is the process of how children learn to resolve correspondences and thereby slowly gain the ability to see sharp images. This includes several procedures such as shape segmentation or the slow increase of resolution of the eyes over months. The for-

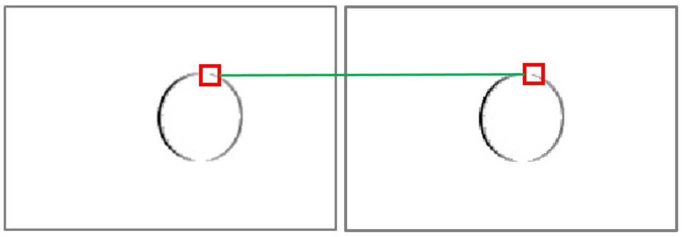


Figure 3.11.: Finding correspondences through shape segmentation, in case of a circle. Image source: (Azanov 2022)

mer refers to the mechanism that the human visual complex can infer correspondences from related forms, as previously discussed in subsection 2.1.3. The latter allows learning initially with reduced complexity, due to a very low resolution, to simplify the search for correspondences. The resolution is gradually increased and thus the complexity of learning. Bootstrapping can be applied to deal with the problem that SOM depend on resolved correspondences during training, as shape segmentation can speed up the learning process. Regarding shape segmentation, the initial SOM implementation, which only recognizes dots in the view fields of both sensors, is extended to recognize 2D shapes. As an easy entry point, exclusively circles are used as shapes, as visualized in Figure 3.11. Circle centers can be recognized by the random sample consensus (RANSAC) algorithm [311] or Hough transformation [312]. The explicit 3D position of the circles is therefore appended to the SOM vector, resulting in $[x_L, y_L, x_R, y_R, r_L, r_R, x, y, z]$. Thereby, x_L, y_L and x_R, y_R refer to the centers of the circle perceived by the left and right sensors and the radius is respectively given by r_L and r_R. This extension does not require an adaptation of the learning algorithm as only the vector size is changed. Furthermore, all points belonging to a recognized circle are reduced to a xy position and a radius of this circle. Subsequently, the circle centers can be used to train SOM without complex calibration.

3.3. Experiments and Results

The transfer of this method from artificially created event-based data to an event camera is rather trivial. However, it is a lot harder to obtain the required information for quality analysis on real hardware. Consequently, in addition to the evaluation on real hardware in subsection 3.3.2, experiments have also been carried out in simulation in subsection 3.3.1. By using ray tracing, enough realism is assumed while having absolute measures to make a qualitative analysis. The sensor placement and calculation of the camera parameters are implemented with the open-source software Blender. Calibration regarding intrinsic and extrinsic camera parameters of both sensors is necessary for simulation. Moreover, to obtain a good error metric, back projecting the SOM's 4D weights to 3D space also requires calibration. However, calibration is neither used for learning nor to solve the correspondence problem.

3.3.1. Simulated Event-based Data

The evaluation of simulated data embodies two parts. Firstly, the performance of learning is evaluated by a qualitative analysis of how a SOM matches the 3D space it was trained in. Secondly, a proof-of-concept is provided that demonstrates that the correspondence problem on event-based data can be solved by a pre-trained SOM.

A Qualitative Analysis

The workspace regarding the target application of a reactive planning system is constrained to $2m^3$ for the experiments within this section. As visualized in Figure 3.7, the SOM discretizes the workspace and thereby creates something like a voxel grid, which was also used within Figure 3.8.

As feature weights are presented in 4D, their error, thus the SOM's error, is difficult to measure. 4D feature weights can be projected back to a 3D space for evaluation, as seen in Figure 3.8. However, an error is also generated by the back projection and it is not trivial to determine how large its proportion is to the total error. Thus, while Figure 3.8 is an informative visualization it does not serve well

# neurons	# samples	run time	mean Cartesian	mean neural	max
10^3	2000	0.2	0.177	0.13	2.24
10^3	5000	0.6	0.204	0.10	3.16
14^3	20000	4.1	0.152	0.11	4.47
14^3	50000	10.4	0.159	0.07	4.47
16^3	100000	28.2	0.149	0.07	4.47
16^3	200000	57.2	0.152	0.06	4.58
16^3	500000	141.4	0.140	0.09	5.00
20^3	100000	49.9	0.151	0.03	5.92
20^3	200000	100.4	0.157	0.03	5.92
20^3	500000	257.6	0.162	0.02	5.92

Table 3.1.: Convergence quality concerning the number of training samples and the number of neurons. Thereby, the Cartesian workspace is referred to as 'Cartesian' while the space induced by the neuron's indices is referred to as 'neural'. The time of a training run is given in seconds. Table source: (Steffen et al. 2019b)

for evaluation purposes. Consequently, in Table 3.1 a convergence quality of the trained SOM concerning the number of training samples and the number of neurons is given. Thereby, it is investigated if the vertices of a grid that is regularly

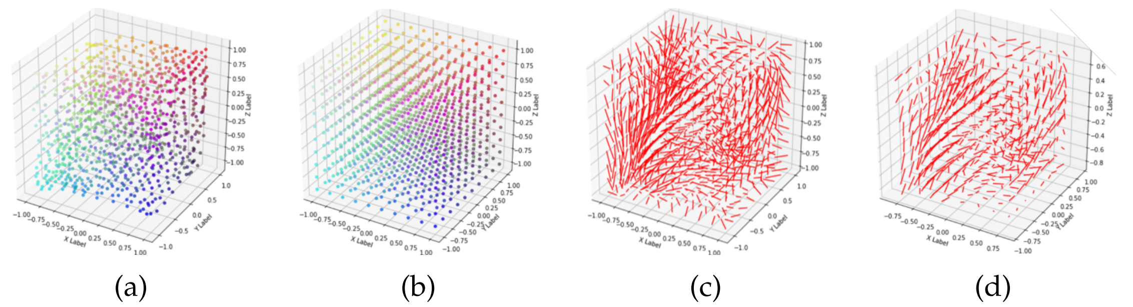


Figure 3.12.: Plots of the root-mean-square error (RMSE), the distance between weight vectors and target representation. (a) Feature weights, (b) target representation (c) RMSE and (d) RMSE but without the neurons on the outer layers. Image source: (Steffen et al. 2019b)

distributed over the workspace can activate the neurons with the same index. This requires prior axis alignment and finding the BMU, as described by the used learning rule, to allow a comparison of the winning neuron and the ground truth. It can be seen nicely in Table 3.1 that the error regarding the space induced by the neuron's indices declines constantly during the upscale of the neuron number and training samples. Even though, the error in the Cartesian space is quite large

the neural mean error is also overall relatively small, proofing convergence and the functionality of the trained SOM. The fact that the error in the Cartesian space is larger does not change that. The table also shows how the run time for training increases for larger networks and more training samples when run on an AMD Ryzen 5 1600x six-core processor with 16 GB RAM.
In Figure 3.12 the RMSE, a measure for differences between predicted model values and experimental results [313], is given for the trained SOM. Thereby, for the RMSE plot in (d), the outer layer was discarded, as SOMs are known to obtain significantly worse results for the rims. As can be seen well by comparing (c) to (d), the average distance is reduced significantly by removing the outer neurons. However, this does not pose a big issue as the workspace for training can be inflated slightly, allowing the removal of the outer neurons.

Solving the Correspondence Problem

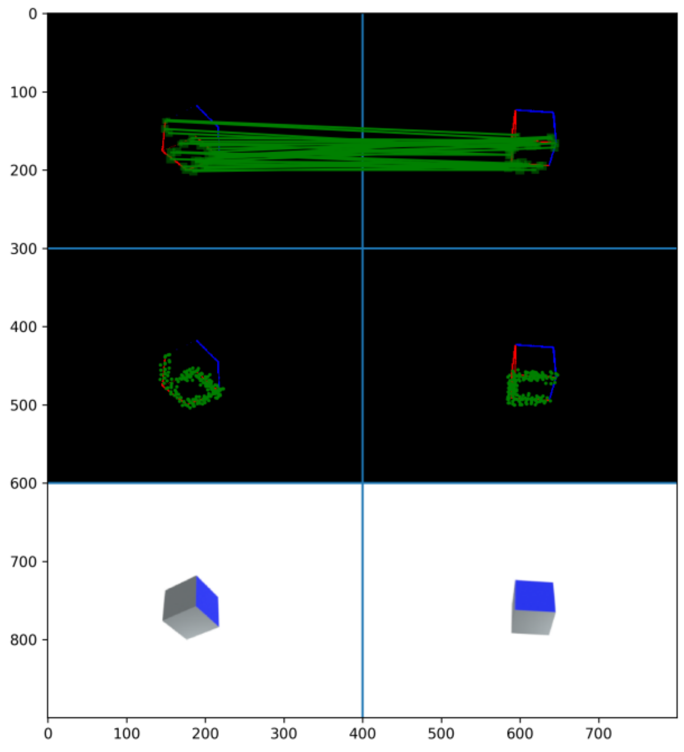


Figure 3.13.: A trained SOM finding correspondences on event-based data. In the first row, every 15th corresponding pair is shown. The center row shows all neurons without connecting lines and the bottom row features the original frames of the simulated object, a moving cube. Thereby, ON-events are marked in red and OFF-events in blue. The green lines connect corresponding neurons and green squares visualize the neurons' regions of activity. Image source: (Steffen et al. 2019b)

This section is a proof-of-concept that a SOM can be successfully applied to solve the correspondence problem as introduced in section 2.1.3. Hereby, a very naive way of matching potential candidates is applied, as all events are matched sequentially with each other, and the polarity is ignored. The process can be accel-

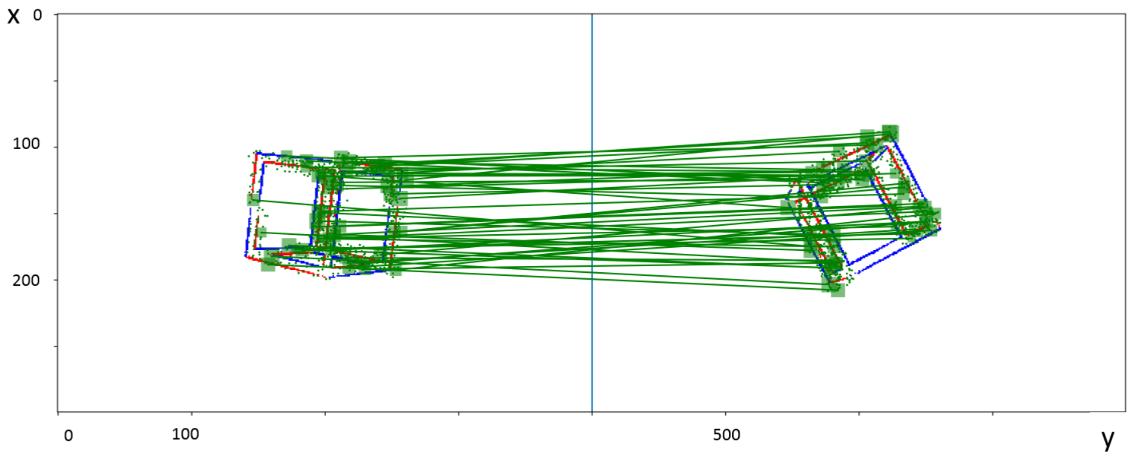


Figure 3.14.: Activity regions of neurons, thus the area of possible matches. Thereby, a wireframe of a rotating cube is recorded in simulation by two event cameras. For an explanation of the elements see Figure 3.13. Image source: (Steffen et al. 2019b)

erated significantly, for example by eliminating candidates through temporary pre-processing. For example, a slowly increasing time interval can be used for this purpose. However, an examination of the algorithm as unadulterated as possible is targeted, thus, run times of 30 s per frame using an $20 \times 20 \times 20$ SOM are tolerated. How correspondences between two slightly shifted perspectives of a moving cube are detected is visualized in Figure 3.13. It is important to note that the sole determination of the BMU is not sufficient to solve the problem of correspondence. Additionally, the distance is used to ensure that potential candidates are located within the activity regions of the respective neurons, thus, the area of possible matches. The size of this region changes with depth. This investigation was also carried out with a more complex object, a wireframe as can be seen in Figure 3.14.

3.3.2. A Stereo Setup of Two Event Cameras

The method is applied to an online demonstrator, as shown in Figure 3.15. Thereby a stereo setup of two event cameras of the type Gen3 ATIS, as introduced in Table 2.4, is mounted approximately 1.2 m above the floor. Specifications about the sensor are given in Table A.1 of Appendix A. The scene to be reconstructed is, in line with the research goal in section 1.2, a moving robot arm, mounted on a table. In (a), a live virtual representation of the sensor setup and the robot in a 3D

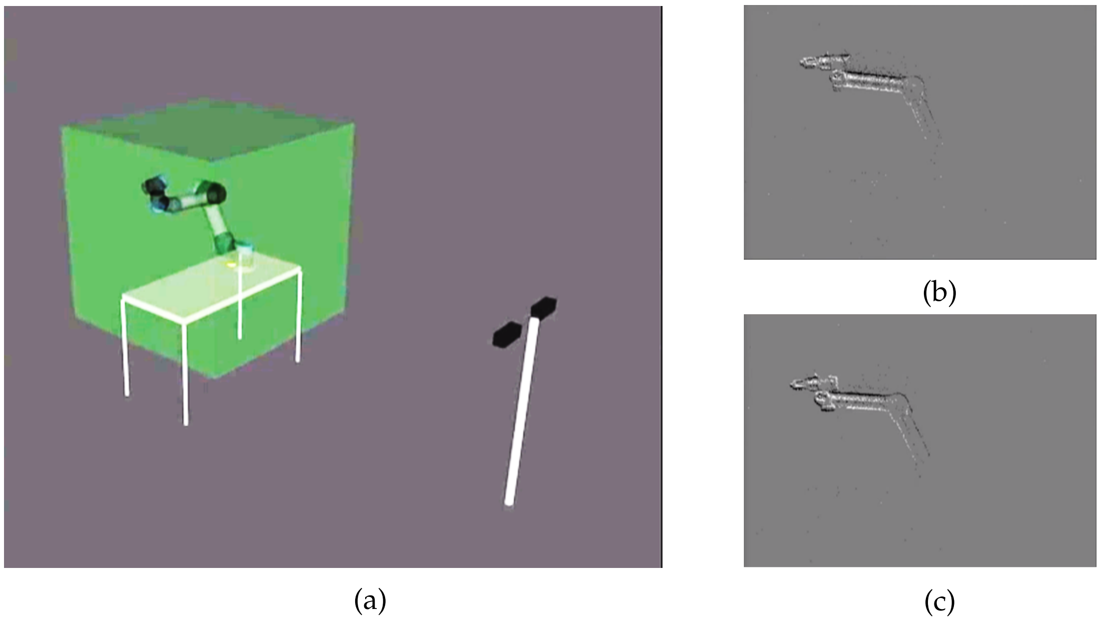


Figure 3.15.: Demonstrator for the execution of the 3D reconstruction on event streams of robot motions. (a) live virtual representation of the sensor setup of two ATIS, the robot and the defined work cell in RViz. Event frames of the scene from the viewpoint of the (b) left and (c) right sensor.

visualization tool for ROS (RViz), is shown. A cube with an edge length of $1.3m$ is defined, as the work cell which covers the robot's motion space. A frame of the event streams, displaying how the scene is perceived by the left and right ATIS, is shown in (b) and (c).

Data Acquisition and Calibration

To extract depth information from a stereo setup, two steps are necessary. Firstly, solving the correspondence problem and secondly, triangulating the obtained correspondences to reconstruct the depth of a real-world point. The triangulation step requires knowledge about the geometrical relations between the two sensors as well as their characteristics. Calibrating a stereo setup, as shown in Figure 3.16(a) & (b), embodies two steps; Firstly a separate mono-calibration of each sensor and secondly the stereo calibration. The first step aims to find intrinsic and extrinsic parameters that describe how the cameras map real-world points to their respective image plane. The second step is necessary to determine how the two sensors are located relative to each other, which is especially important for stereo depth estimation as discussed in subsection 3.1.2. As the correct calibration of event cameras is crucial and at the same time quite complex, many methods and best-practice advice have been published on the subject in the last few years [314; 315; 316; 317]. Traditional calibration methods for frame-based sensors apply a so-called calibration pattern with known geometry. Usually, a

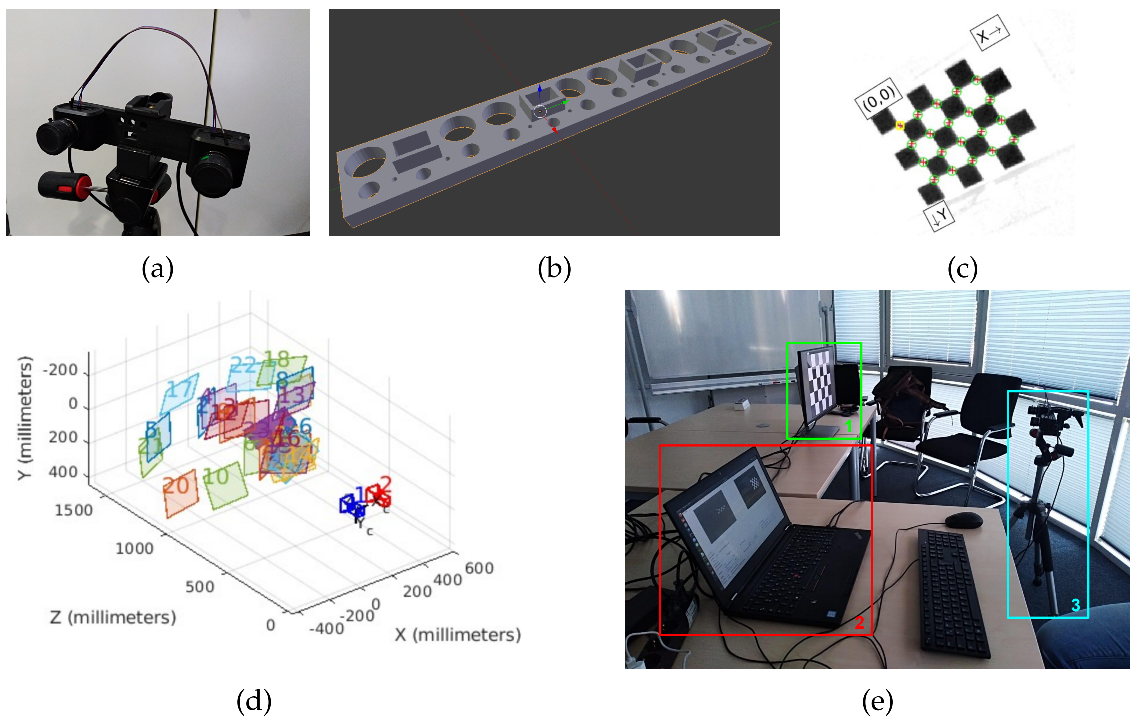
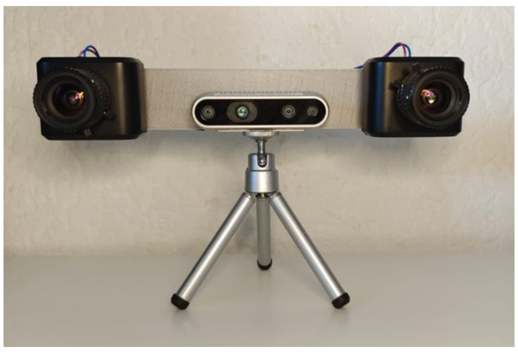


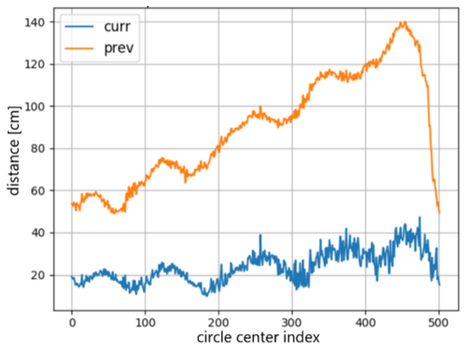
Figure 3.16.: Hardware setup for mono and stereo calibration of an ATIS. (a) Two ATIS fixed on a 3D-printed sensor mount. The three-colored cables, on top of the setup, connect the clock pins, necessary for synchronization. (b) Design of the 3D-printed sensor mount. (c) Reconstructed frame of the flickering checkerboard. Additionally, the detected pattern points (green circles), the coordinate systems for the checkerboard and the re-projected points after the calibration process are shown. (d): Visualization of different perspectives of the checkerboard. (e) Photo is taken during the calibration process. A static monitor displays a GIF of the flickering checkerboard (highlighted in green). The sensor setup's position is altered for each pair of images (highlighted in green). The calibration software is running on a connected laptop (highlighted in red).

checkerboard pattern is used for this purpose. There are, however, other possibilities with more complex geometries. The core of the calibration process is the re-identification of these features in each image, thus the detection of checkerboard corners. Subsequently, an optimization problem is solved to optimize the intrinsic as well as extrinsic parameters [316]. For a printed checkerboard pattern, the known image points will be co-planar and therefore not sufficient to define a 3D space. Hence, the checkerboard needs to be recorded from different viewpoints as shown in Figure 3.16(d). In the special case of event-based sensors, the calibration process becomes slightly more complex, as the sensors only react to changes in illumination and are not able to detect a stationary checkerboard. Thus, due to the asynchronous mode of action of event cameras, this technique is not directly applicable. Hence, to transfer the technique to event cameras, either the camera or the pattern must move [317]. Therefore, many methods for calibrating event cameras use actively illuminated patterns. Either a custom-built LED array [315] or electronically displayed flashing checkerboards [318; 319]. Both methods allow feature detection due to the fast illumination changes and subsequently enable the use of standard optimization-based calibration tools. A blinking LED board implements the possibility of designing a calibration pattern with non-co-planar points. However, it has the disadvantage that they cannot be reliably detected by frame-based cameras, which excludes them for use with a multi-sensor setup that includes event-based and frame-based cameras as used in (Elfgen 2020; Steffen et al. 2021a). Furthermore, due to the need for precision, the production of such boards is neither easy nor cheap. Faulty boards lead to poor calibration results which can severely affect the stereo reconstruction. Quite recently, alternatives like ANN-based image reconstruction [316; 320] have been proposed, which do not need actively illuminated devices. Furthermore, in [317] calibration is performed directly on events that are triggered by relative motions between the pattern and sensors.

Verification of the Calibration by Using Groundtruth



(a)



(b)

Figure 3.17.: Validation of the calibration result through sensor fusion. (a) The multi-sensor setup embodies two ATIS and an Intel RealSense D435. (b) Calibration error over time for Figure 3.18(a) & (c) in orange and for Figure 3.18(b) & (d) in blue. Image source: (Azanov 2022)

A successful application of any method for 3D reconstruction requires a suitable, accurate calibration of the sensors. In this case, this concerns the mono calibration of both ATIS and their stereo calibration. The calibration process of event cameras is error-prone and in addition, their finished calibration is not very robust. To overcome this issue a method was developed to validate the calibration result using sensor fusion and shape segmentation. By using the RealSense RGB ground truth, 3D points that are suitable for comparison with the results of the calibration, are generated. Specifications about the depth sensor are given in Table A.2 of Appendix A. As an example, a bad calibration of event-based sensors is compared with a good one in Figure 3.18. For this purpose, shape segmentation

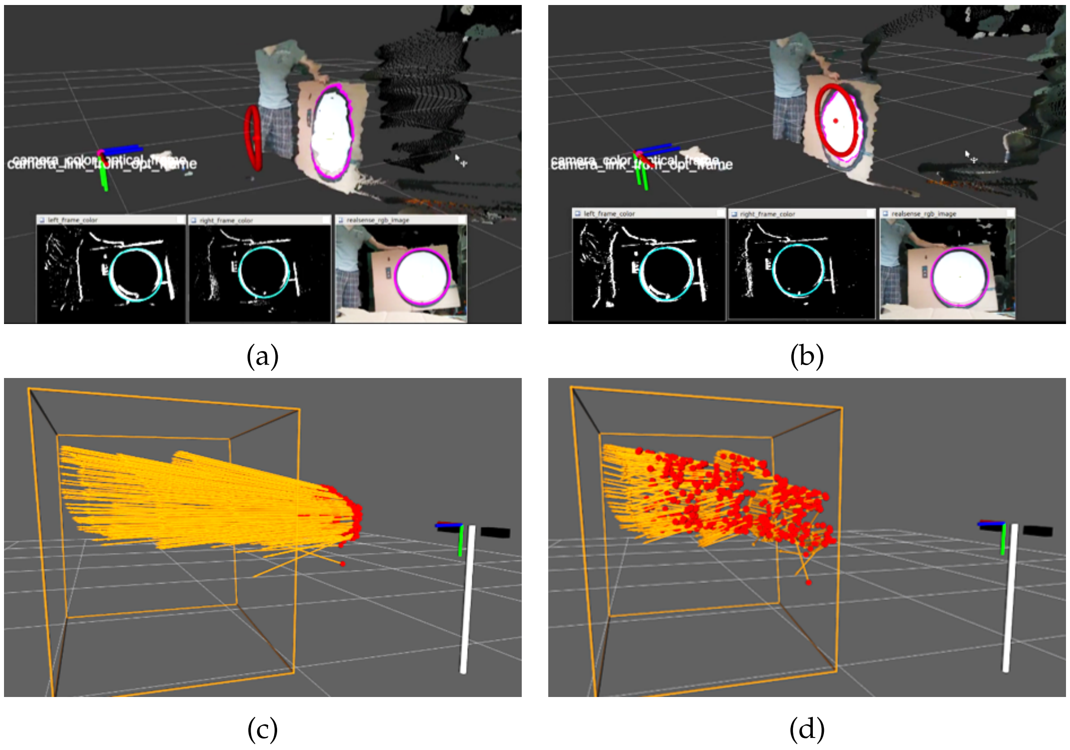


Figure 3.18.: Investigation of calibration quality using sensor fusion. (a) & (c) poor calibration results of event-based sensors compared with (b) & (d) significantly better ones. In (a) & (b), the difference in the quality of the calibration can be seen by the distance of the two detected circles. In (c) & (d), the error is visualized by orange arrows, pointing from the triangulated circle centers acquired with RANSAC on event streams to the circle centers received by the Hough transformation on data from the depth sensor. Image source: (Azanov 2022)

is used, more precisely, circles are compared. In the case of event-based sensors, circles are detected by a RANSAC algorithm [311] and for RGB data through a Hough transformation [312], as visualized in Figure 3.18(a) & (b). The circles in the event-based case are not so easy to see with a Hough transformation, as there are not always enough events to form a complete circle. Thereby, the red circle is triangulated with different calibrations. It is visible that the calibration in (b) is superior. Additionally, in Figure 3.18(c) & (d) 502 circles are detected and plotted. The arrows point from the triangulated positions of circle centers, obtained by event-based data, to the 3D positions acquired using a RealSense. The lengths of these arrows, representing the calibration error, are plotted over time in Figure 3.17(b), in blue for (d) and in orange for (c) respectively. The error is the absolute distance of the circle centers, hence, the calibration quality is well-measurable.

Evaluation of the Pre-learning Process

To allow a transfer of the proposed method from simulation to event streams of the ATIS, the SOM is pre-trained on a geometry that corresponds exactly to the dimensions of the intended workspace. The workspace is visualized in Figure 3.15,

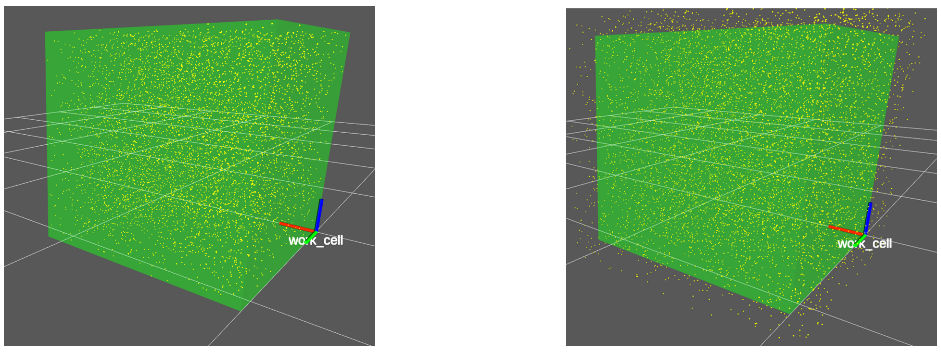


Figure 3.19.: Training samples were drawn from a cubic space, corresponding to the dimensions of the intended workspace of the application. In (a), it is only sampled in the vicinity of the workspace, while in (b), training samples that are located up to 10 cm outside the cell are included. Image source: (Azanov 2022)

except that the cameras and the robot have been neglected for visualization purposes. In Figure 3.19, the workspace and the generated training examples are visualized. However, SOM are known to perform poorly regarding their outer

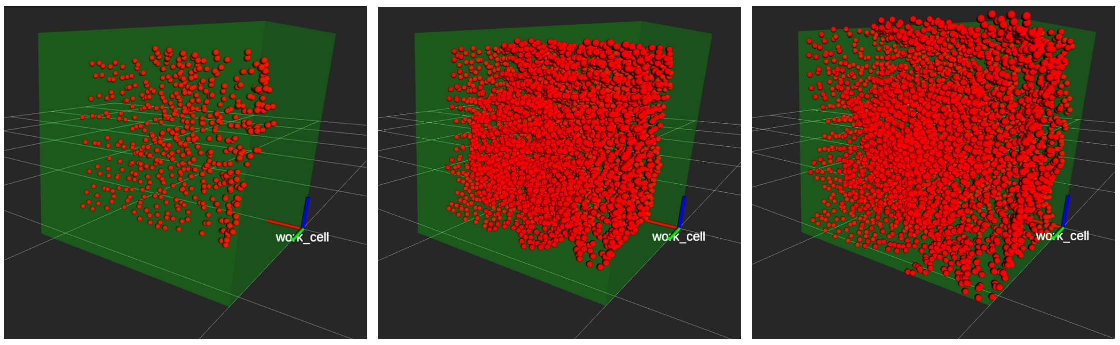


Figure 3.20.: Evaluation of a pre-learned SOM. A SOM with size $8 \times 8 \times 8$ is trained in (a), while a SOM with size $16 \times 16 \times 16$ in (b) & (c). Samples are drawn from within the work cell for (a) & (b) and up to 10 cm outside the workspace for (c). Image source: (Azanov 2022)

layers, as discussed in subsection 3.2.3. Thus, in addition to creating the training examples within the borders of the workspace as seen in Figure 3.19(a), in (b) the workspace was extended by 10 cm in all directions. The size of a SOM required to represent a workspace well must be tested, as visualized in Figure 3.20. It can be seen that the SOM of size $16 \times 16 \times 16$ depict the workspace much better than the SOM of size $8 \times 8 \times 8$. All SOMs in Figure 3.20 are trained with 10 k samples, whereby the samples are exclusively drawn from within the work cell for (a) & (b), as shown in Figure 3.19(a). In contrast for (c), samples are drawn up to 10 cm outside the workspace, as visualized in Figure 3.19(b). As expected, due to the weakness of SOMs concerning their edge regions, a much better representation of the work cell is produced when it is slightly inflated before sampling.

Evaluation of Receptive Fields for Local Matching

The approach with the improvements as described in subsection 3.2.4 was evaluated on simulated data. In this way, errors and inaccuracies that inevitably arise from the sensor calibration are prevented. In Figure 3.21 results of the evaluation

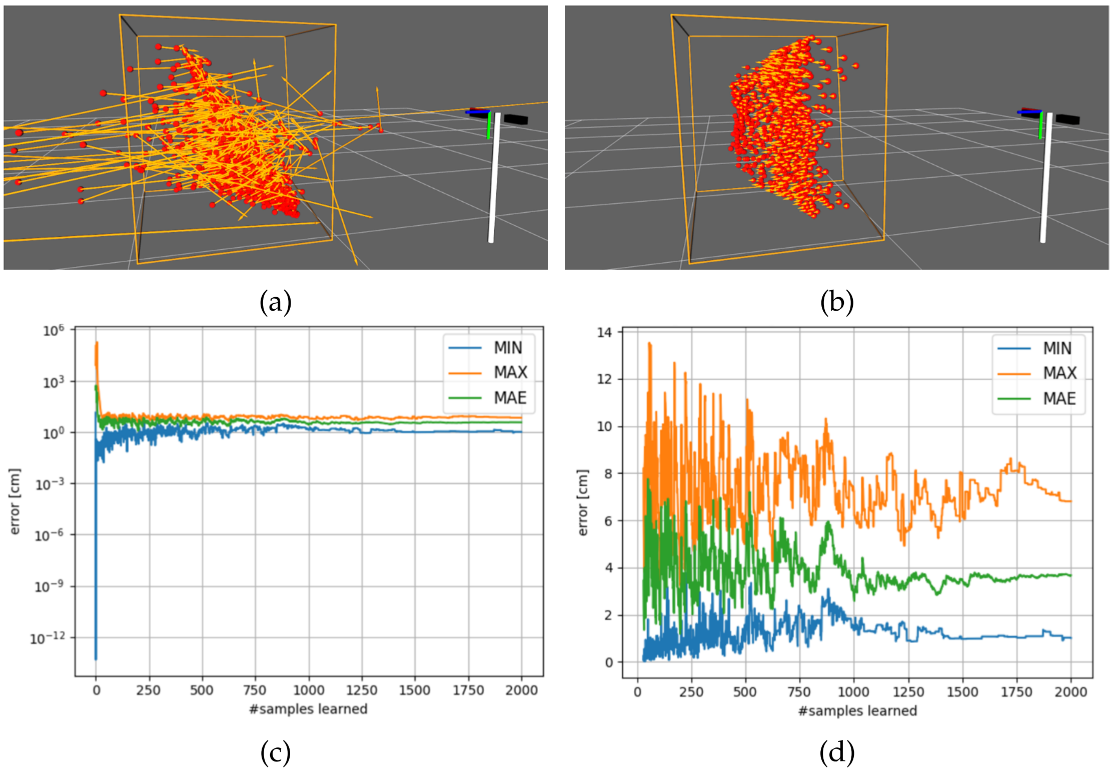


Figure 3.21.: Evaluation of triangulated 3D positions by a SOM from simulated event-based data. The arrows indicate the difference between reconstructed 3D positions and ground truth at the (a) start and (b) after a completed learning process. (d) How the error decreases during learning is plotted in (c), supplemented with a zoom in (d). Image source: (Azanov 2022)

in simulation can be seen. The neurons used for this purpose have a combination of 4D and 3D values, $[x_L, y_L, x_R, y_R, x, y, z]$. The network is trained with samples of this data format, whereby the first four values indicate the pixel coordinates on both event cameras and the last three indicate the ground truth. This is visualized in Figure 3.21(a) & (b), where the arrows indicate the difference between the reconstructed positions by the SOM and the ground truth. In (a) the SOM was trained with only 10 samples beforehand thus, the graphic represents the situation before learning, or very much at the beginning. Since the initialization of the

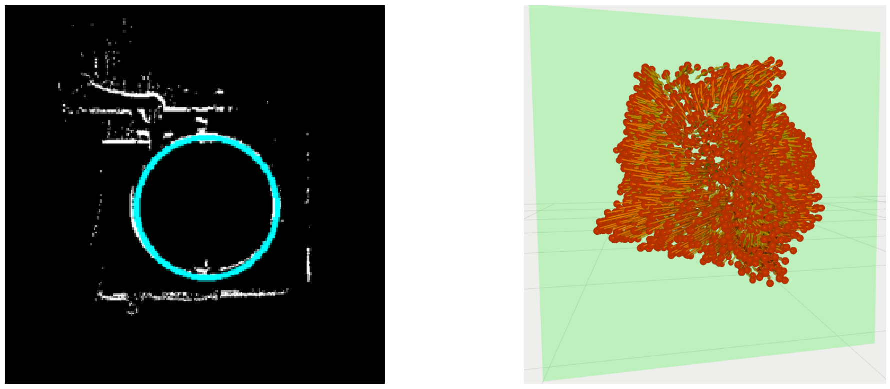

(a) (b)

Figure 3.22.: Sensor fusion for the evaluation of triangulated 3D positions by a SOM from event-based data. (a) Two-color image of the ATIS where the reconstructed circle is marked in cyan. (b) Visualization of a $15 \times 15 \times 15$ SOM after learning, the orange arrows indicate the error, as the distance of the triangulated values to the ground truth. Image source: (Azanov 2022)

SOM's weights is random, the initial state is quite chaotic. In contrast, in (b) 2000 samples have been presented to the SOM, so this is a post-learning shot. How the error develops during learning is plotted in Figure 3.21(c) and Figure 3.21(d). It can be seen that the SOM converges after only 500 training cycles and afterward the mean error stays below 4 cm. The distances between triangulated and ground truth 3D positions were used as the error measure, thereby, the maximum, minimum and mean value is given. The graph in Figure 3.21(c) shows the whole process and (d) a zoom-in where the first 30 samples are skipped. It can be seen that the mean error, visualized in green, remains below 4 cm in the last half of the training time. The large error at the beginning implies unfavorable random initialization of the neuron weights, which is however the norm.
The ground truth obtained by sensor fusion can also be used for an evaluation of the stereo reconstruction on real event-based data, as shown in Figure 3.22. A circle was chosen as the object to be shown to the cameras, as seen in Figure 3.22(a) perceived by the ATIS. For training a SOM with the lattice size $15 \times 15 \times 15$ is used, as shown in Figure 3.22(b). Subsequently, circles are detected in the event-based data by a RANSAC algorithm and for data of the RealSense through a Hough transformation. The error is the distance of the circles which resulted from the reconstruction of the respective sensor data. Meaning, the triangulated 3D positions generated by the SOM from event-based data are compared with

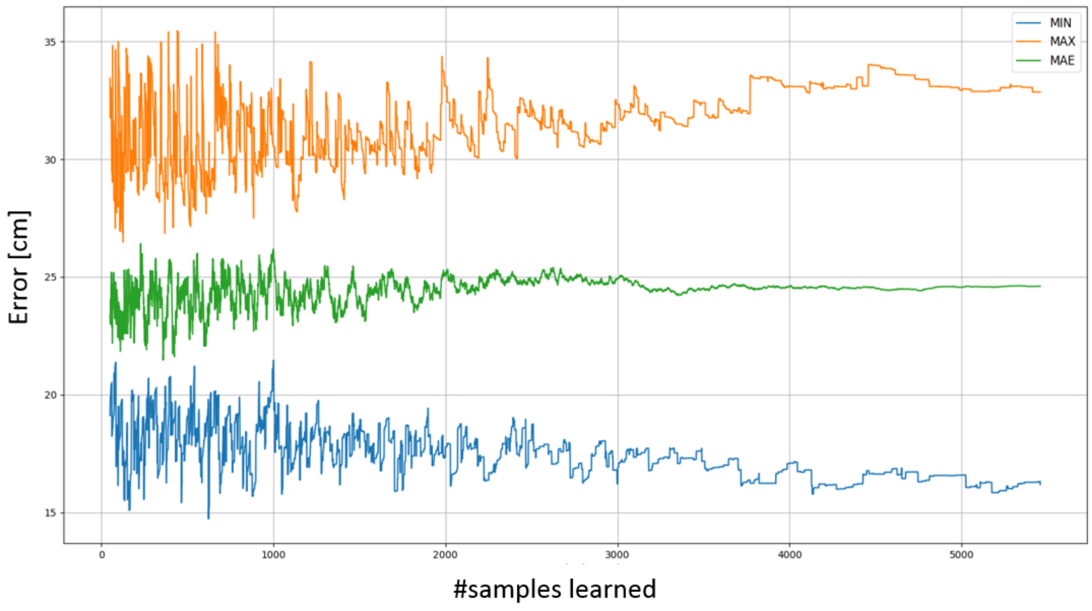


Figure 3.23.: Evaluation results obtained by using sensor fusion. The plot visualizes the error regarding the 3D reconstruction and how it evolves, thereby the maximum, minimum and average of the error are visualized. Image source: (Azanov 2022)

the ground truth obtained from the RealSense. The calibration of the combined sensor setup, shown in Figure 3.17(a), is difficult, which harms the results. Samples from RealSense differ strongly from the triangulated values, as can be seen in Figure 3.22(b) and the error is correspondingly large. The results are plotted in Figure 3.22. The plot in Figure 3.23 shows that the evaluation on real data is far behind the results obtained in simulation, the difference between the results is almost 20 cm.

3.4. Discussion

As state-of-the-art is quite thin, at least concerning algorithms dealing with both, stereo reconstruction and event cameras, a more profound review of related work is provided for this chapter. Furthermore, this chapter is very exploratory, as a new method was developed. This method was tested both in simulation and on real event-based data, which could prove its general suitability. As it dealt with a relatively new technology, event cameras, new ground was broken. It was shown that the method works, but especially in terms of accuracy, it is not directly concordant with stereo vision methods regarding classical computer vision. In comparison to similar work based on self-organization [190; 189], no feature matching was done here. In contrast, a neural representation whereby each node corresponds to a 3D occupancy region is created. A powerful feature of the approach is that the 3D reconstruction implicitly contains extrinsic and intrinsic

camera calibration. These parameters as well as the lens distortion are learned implicitly by the SOM. Moreover, event-based techniques for depth reconstruction often embody complex complementary constraints [55; 258; 54]. In contrast, the system presented here differs regarding its simplicity. The accuracy of the depth reconstruction is highly dependent on the size of the neural grid, which grows with $\mathcal{O}(n^3)$. The network's size and complexity are therefore not exponential but only cubic. Nevertheless, it causes the classical trade-off between speed and memory consumption. A workspace of $1m^3$ and a voxel size of 5 cm, however, is considered sufficient for the intended use case, presented in section 1.2, which generates a network of size $20 \times 20 \times 20$ with the reasonable amount of 8 000 neurons. As a consequence, the relatively small voxel grid enables the method to be a comparatively memory-friendly solution, despite the relatively high computational complexity.
Biological plausibility, which explains 3D reconstruction in the cortex, is a very desirable property of a stereo vision method. However, even though biologically inspired methods, such as event cameras (see subsection 2.2.3) and SOM (see subsection 2.1.2) are used, the presented approach does not claim to be biologically plausible. To some extent, the algorithm presented can be considered to generalize well. It has been used here for event-based data, but adaptation to RGB cameras would only require a prior color or feature matching. However, this is only a theoretical consideration and has not been tested. The extensions, see subsection 3.2.4 and 3.2.3, already represent an enormous improvement of the algorithm. However, many possibilities to optimize the processing further, both in terms of speed and accuracy, are still possible. An interesting example, in this respect, is the employment of sparse matrices [321] either together with receptive fields or even as a substitute. Moreover, a more sophisticated method than inflating the workspace, as presented in subsection 3.2.3, could be developed to ensure correct learning of the rims.
A major problem, however, is the relatively poor results on the event-based data in Figure 3.23 compared to the evaluation in simulation in Figure 3.21. While the results in simulation show only a relatively small error after training, a relatively large error remains when using real data. One reason for this are remaining calibration problems, as can be seen in Figure 3.18. It is further assumed that the ground truth obtained with the multi-sensor setup, shown in Figure 3.17, also induces further inaccuracies. It is not easy to evaluate the 3D reconstruction on real data, therefore in the previous chapter, a lot of the experiments were done with simulated data.
During this thesis, it became clear that a stereo setup of two event cameras is not sufficient to capture the work cell shown in Figure 1.1. Furthermore, event cameras are comparatively expensive. Thus, the integration of additional ones would be a costly, albeit interesting, undertaking. In the case of sensor fusion, integrating a conventional camera into the system, the cost factor could be circumvented. However, the implementation involves an enormous complexity. In addition, the correlation between resolution and network size also posed a certain problem for practical use. Therefore, an existing vision system was used in chapter 4, to not influence the results.

4. Reactive Neural Path Planning

As the role of Human–robot Interaction (HRI) constantly and drastically increases, robots need to act safely and flexibly in a shared workspace with a human. To enable a safe and cooperative collaboration, it is necessary to perform a goal-directed motion while taking a dynamically changing environment into account. This requires reactive path planning with dynamic obstacle avoidance [322; 323; 324]. There are two fundamentally different approaches for robotic motion planning [325; 326]. First, in the task space T, requiring Inverse Kinematic (IK) for the calculation of joint angles afterward. Second, in the N-dimensional configuration space (C-space), where N is the Degree of freedom (DOF). While planning in the task space is significantly less computationally intensive, it faces redundancies and may even lead to unpleasant joint angle jumps [18; 17]. Motion planning in

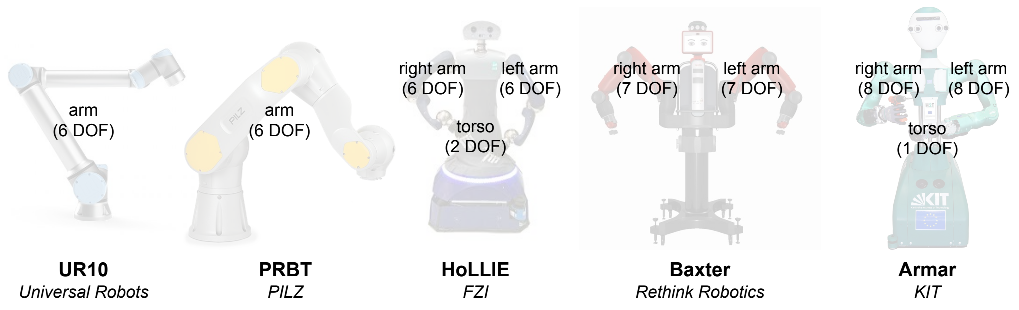


Figure 4.1.: Overview of the DOF of industrial robot arms and humanoids. The DOF of the neck, hands and mobile platforms are not regarded for collision-free arm movements. *Data and images: UR10* [1], *PRBT* [2], *HoLLiE [327], Baxter [328], Armar [329].*

the high-dimensional C-space is more elegant as it becomes a pure path planning problem. The robot is represented as a point, which has benefits, especially for obstacle avoidance. Many techniques for path planning, like the grid-based search A* [28], bug algorithms [330], visibility graphs [331], Voronoi diagrams [332; 333] and cell decompositions [322; 326] have been proposed. They are all complete and work well in lower dimensions and with static obstacles [334], however, they do not scale well to high dimensions [322; 323; 326]. HRI requires real-time planning to ensure the safety of humans. Related algorithms are still not performant enough, especially in unpredictable surroundings [335] and for robots with many DOF. An exemplary representation of a typical number of DOF of robots in a scientific and industrial context is given in Figure 4.1. This chapter addresses re-

search question 2 defined in chapter 1: *"Is it feasible to use the high-dimensional configuration space, which requires huge amounts of neurons, for neural path planning?"*.
The remainder of this chapter is structured as follows. The relevant related work is covered in section 4.1. The theoretical basis of the developed method is given in section 4.2. First, the initial idea is presented in subsection 4.2.1, the reduction of the C-space to allow fast and efficient path planning. Then, different network models are introduced which are considered for the reduction of the complexity in subsection 4.2.2. Subsequently, it is depicted how obstacles can be transformed into the reduced subspace in subsection 4.2.3 which is concluded by path planning in a cognitive map in subsection 4.2.4. An evaluation of the presented method is shown in section 4.3, starting with a comprehensive comparison of network models in subsection 4.3.1. Different algorithms for subsequent path planning are tested in subsection 4.3.2 and obstacle avoidance of the most suited network models is tested in subsection 4.3.3. Finally, the most important component of the evaluation is shown in subsection 4.3.4, a comparison of the presented approach to modern sample-based planners. The chapter is completed with an in-depth discussion in section 4.4. The material covered in this chapter was originally published by the author in (Steffen et al. 2021c; Steffen et al. 2022a; Steffen et al. 2022b).

4.1. State-of-the-art

Path planning in robotics is by no means a new problem, so the respective state-of-the-art is very extensive, an overview of the most important methods is given in subsection 4.1.1. In subsection 2.1.4 an introduction was given to how the brain solves spatial navigation and path finding through extreme parallelization. Brain-inspired methods for path planning, based on these neurological findings, are introduced in subsection 4.1.2. In the course of this thesis, my neural implementation for a 3D Wavefront Algorithm (WFA) was developed (Steffen et al. 2020b). As it is well suited for execution on parallel hardware it serves as a basis for the respective performance analysis presented in chapter 5, and is therefore introduced in subsection 5.2.1. However, motion planning, computing a collision-free path from a start to a goal configuration in a dynamic environment requires very fast and flexible planning algorithms. As the methods presented in subsection 4.1.1 were mostly developed for 2D, or sometimes 3D, they are not well suited for planning in the C-space of a robot with many DOF. The same is true for many of the neural applications in subsection 4.1.2. The implementation introduced in (Steffen et al. 2020b) was not performant enough for an online demonstrator for highly reactive path planning. Furthermore, only the end-effector was considered here, thus, the joint angle position of the robot is not reliably determined by the algorithm, which poses a safety problem for the intended application. As a result, a biologically inspired method has been developed that reduces the computational burden by dimensionality reduction and efficient C-space creation. Related work regarding these topics is given in subsection 4.1.3.

4.1.1. Conventional Methods for Path and Motion Planning

Research on robot motion planning started in the ’60s, however, it was not until the end of the ’70s with the work of Lozano-Pérez’s [331] that it received greater attention. As path planning algorithms are often applied for robotic arm motion control this section also covers 2D and 3D navigation. In this context path planning is regarded as the association of a system’s start to its goal configuration [336]. Comprehensive surveys on path planning algorithms for mobile robots were published relatively recently in [337; 336]. A corresponding work regarding 3D navigation is [338]. However, as the planning complexity increases with the number of DOF [336], many of the planning algorithms that work quickly and reliably in 2D are very slow to the point of unusability in higher dimensional search spaces. In addition, both robots and the environments they operate in, have become much more complex. This increases further the need for efficient high-dimensional motion planning [334]. In particular, dynamic environments that require constant replanning place high demands on processing times. A review that is unfortunately not quite as up to date but has the focus on actual motion planning in the C-space was published in [339]. The algorithms are thereby divided into classic and heuristic methods, including analysis of their frequency of use over time. Until 1982 only classical approaches were used, however, since 1993 more than 50 % robotic applications use heuristic techniques. A more recent paper regarding planning in high-dimensional spaces was presented in [334]. In general, path planning algorithms are often divided into categories, *(1) grid based*, *(2) sampling based*, *(3) Artificial Potential Field (APF)*, *(4) mathematical optimization models* and *(5) bio-inspired algorithms*. However, slightly different divisions are known in the literature [338; 334; 339].
Category *(1)* is either referred to as *grid based* [334] or *node based optimal* [338]. Here, the search space is divided into discrete cells that are either occupied by an obstacle or free. The shortest path that does not collide with any occupied cells is to be found between the cells defined as start and target. A very straightforward method, which since the ’90s is becoming increasingly widespread [340; 341], is the WFA, also called grassfire. It uses a Breadth-first Search (BFS), an uninformed search algorithm, which is used to traverse the nodes of a graph. All nodes that can be reached from the current node are always considered first, instead of descending directly into the depths. The algorithm encodes the distance of any map location regarding the start cell, called the source, through propagating waves through the network. Thereby, all cells with the same distance to the source are passed synchronously and the designated values, with which the cells are marked, increase with each wave. To determine the shortest path from any point on the map to the source the highest descent of wavefronts is traced back [24; 342]. The BFS applied for the WFA is formalized in algorithm 1. The

Algorithm 1 Breadth First Search adapted from: [29]

```
procedure BFS($G$ : graph, $s$: start vertex)
    $Q$ : queue
    $Q.enqueue(s)$
    mark $s$ as visited
    $parent$ dictionary                                  ▷ node → parent node
    $parent[s] \leftarrow s$
    while $Q$ not empty do
        $v = Q.dequeue()$
        for all neighbors $w$ of $v$ in $G$ do
            if $w$ is not visited then $Q.enqueue(w)$
                $parent[w] \leftarrow v$
                mark $w$ as visited
            end if
        end for
    end while
end procedure
```

algorithm is very simple, efficient and resource-saving. Furthermore, it is not affected by complex shaped objects and only to a very small extent by the map resolution [24]. Improved versions that avoid unnecessary explorations, thus increasing the efficiency, have been presented in [343] with the focused WFA and in [344] with the optimally focused WFA. However, this method only generates a near-optimal path regarding distance and traversal costs [47], while Dijkstra's algorithm and its famous derivative the A* are optimal. Furthermore, the algorithm uses a graph search that does not take edge weights into account, which is a disadvantage for some applications. In contrast, Dijkstra's and the A* represent the workspace with a graph of weighted edges between adjacent cells, by computing a value at each node with an estimator function. The weights indicate specific travel costs from one cell to another. In a way, the WFA can be seen as a special case of Dijkstra's algorithm with equal costs for all edges. How this affects a practical path planning application is visualized in Figure 4.2. In terms of searching the shortest path in a graph, Dijkstra's algorithm is seen not only as a classic representative but also as the most mature one [31]. However, it is quite time-consuming and memory-consuming [31]. The A*, developed in 1968 by Peter Hart et al. [28], reduces the time complexity of the search through an additional heuristic function to the traversing costs, which leads more efficiently to the goal. In [32], an efficient A* algorithm is presented. The authors claim to achieve a reduction in computation time of up to 95 %. This tremendous progress is achieved by computing the heuristic function for a node not at the beginning but just before a collision. This avoids many unnecessary computations which has a positive impact on performance. An extension specialized for handling dynamic obstacles was introduced as the D* [30]. It was designed to create collision-free paths in dynamically changing surroundings. This informed incre-

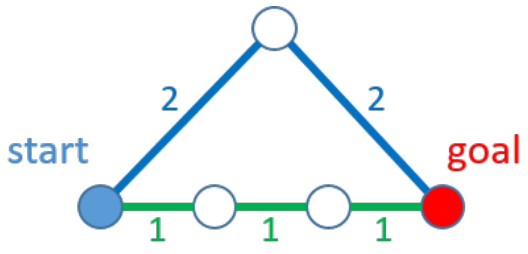


Figure 4.2.: Even though the shortest path from start to goal is the green one when considering edge weight, the WFA that ignores weighted edges, would choose the blue path. As Dijkstra's algorithm takes edge weights into account it would select the green path as the shortest one. Image source: (Weyer 2021)

mental search algorithm massively reduces calculation times, but also has major problems in terms of storage space [336]. Grid-based methods are complete and optimal, thus the shortest path is found if there is one. However, grid-based techniques have high computational complexity, therefore even the newest most sophisticated extensions suffer from the curse of dimensionality, which makes them unsuitable for systems with many DOF.

The second category *(2) sampling based* algorithms sample the space randomly, whereby they check for each sample if it is in the free space and connect new samples to existing ones. The Probabilistic Road Map (PRM) [19] samples the space evenly and subsequently uses a grid-based method for pathfinding. In contrast, the Rapidly Exploring Random Tree (RRT) is a sampling-based extension of the A*, complementing it by using a Voronoi bias to divide the search space into uniform regions. In addition, the RRT allows the graph to be expanded at any point. Its founder, LaValle [20], calls it a randomized data structure designed for a wide range of path planning problems but particularly good at handling non-holonomic constraints and many DOF. The RRT is preferable to the A* in the case of a high-dimensional search space. Many optimization of the RRT exist [345; 33; 34]. A well-known representative is the Rapidly Exploring Random Tree Connect (RRT-C) [34]. It uses two incremental RRTs that are initialized at the start and destination points of the motion, respectively. Dual search can significantly increase performance which the authors demonstrate using a PUMA robot arm with 6 DOF as an example. In general, as sampling-based techniques are more efficient than grid-based methods they are better suited for motion planning for robots with many DOF. But, as they are only probabilistically complete and cannot determine if no solution exists they offer weaker guarantees than grid-based methods and often produce only sub-optimal solutions [334]. Also, online applications with narrow passages pose a problem as probabilistic completeness also implies that an existing path is eventually found for, however, under circumstances only for infinite time [346]. Furthermore, this means that the resulting paths may vary for different executions with identical circumstances, as sampling-based methods are not deterministic. Nonetheless, due to their great

efficiency, sampling-based algorithms are now mainstream for motion control with robots with many DOF [347]. A comprehensive survey for sample-based methods for robot motion control is provided in [348]. In [349] a more recent benchmark is given, however, the authors focus on outdoor navigation.
Within category *(3) APF* [350], the robot is represented as a point, a sort of particle that is under the influence of a potential field. Thereby, two forces act on the robot, an attracting potential that goes out from the goal and a repelling potential from obstacles [322; 339]. APF are fast and effective but might lead to sub-optimal solutions as they may get stuck in local minima, especially for cluttered maps.
For category *(4) mathematical optimization models*, factors like time, acceleration profile and energy effort are optimized by use of differential equations and dynamic constraints. Thereby, finding a path between the start and end configuration is interpreted as a mathematical optimization problem [339]. Even though, these methods handle dynamical obstacles well, being complex they suffer from high computational costs and are thus only suitable for off-line applications [338].
Lastly, category *(5) bio-inspired algorithms*, is often divided into evolutionary algorithms and Artificial Neural Network (ANN). Evolutionary algorithms optimize a given path through an evolutionary process whereby new generations are generated with random mutations and a fitness function steers the development. It handles complex and dynamically unstructured constraints very well but suffers from slow convergence which makes it unfit for online applications. As the field of research regarding motion control with ANN is highly topical and very comprehensive, it will be dealt with in greater depth in subsection 4.1.2.

4.1.2. Brain-inspired Methods for Path and Motion Planning

Methods using ANN are fundamentally similar to category *(3)* [338]. They use their neural space to represent a robot's C-space [339]. The neural network enables real-time planning as motions are generated online by the network's dynamics. This is possible as neither prior knowledge, local collision checks in the task space nor optimizing a cost function is necessary [339]. Which makes this method very efficient and computationally lightweight.

Deep Learning for Motion Planning

Many recent methods for path planning with ANN for robots with many DOF in the C-space combine conventional methods with deep learning [351; 352; 353; 354; 355; 356]. In [356] a method to tackle the IK problem with an ANN is introduced. Thereby, Cartesian coordinates of the end-effector are used as training input and joint angles are generated as output. This method seems to be more sophisticated than analytic IK solutions. In [355] an approach for 3D navigation of the end-effector using APF is presented. The authors provide proof of the real-time capability of their algorithm, however, detours are still sometimes generated due to local minima. The proposed Motion Planning Network in [351] embodies two networks, one for encoding and a feed-forward multi-layer network for planning. The former generates offline a feature space representing static obstacles out of a point cloud. The latter uses the feature space, a goal configuration and the robot's current joint angles as input and performs online path planning. The authors include the RRT to their framework and thus achieve probabilistic completeness. But, this goes hand in hand with the withdrawal of conventional path planning algorithms like slow convergence in high-dimensional spaces [352]. One problem with many of these approaches [351; 352; 353; 354], however, is that they work with known maps and a fixed goal. Therefore, reactive, flexible motion planning in a dynamically changing environment is not considered. Nevertheless, the method in [356] is evaluated on a 5 DOF and in [355; 352] on a 7 DOF robot arm. Furthermore, the real-time capability is explicitly claimed in [352; 355]. Several methods for path planning based on Reinforcement Learning (RL) have been proposed [357; 358; 359; 360], as RL lends itself to the task due to its good generalization properties. However, it has been shown that these approaches are very difficult to train and also require great simplifications [361]. That RL is not well suited for this task was also confirmed in [362]. The authors report that they could not achieve any good results even with great effort concerning parameter tuning, architecture, and reward signal and have therefore turned away thematically from RL. Moreover, most of the proposed techniques are targeted at the navigation of mobile robots [359; 358; 360; 362].

Planning with SNN for Mobile Robots

Due to their optimality and completeness grid-based methods, presented in subsection 4.1.1, are superior to other methods in terms of the quality of the generated path. However, they suffer strongly from the curse of dimensionality, which is why they are not suitable for high-dimensional search spaces, at least not on conventional hardware. Developments in the field of Spiking Neural Network (SNN) (subsection 2.2.1) and neuromorphic hardware (section 2.2.1) open up new possibilities to apply optimal planners in higher dimensions through external parallelization. By imitating sparsity and parallel asynchronous computation known from the brain, these technologies have proven to have the potential for many real-time applications [363] like, stereo vision [294], real-time evaluation of medical data [364], energy efficient unidimensional Simultaneous Localization and Mapping (SLAM) [365] and keyword spotting [366]. An especially good fit for spiking and neuromorphic applications in terms of planning algorithms is the WFA [47; 49], due to utilizing massive parallelization. A common feature of many methods presented in this section is that they are based on the WFA and are strongly inspired by navigation techniques of the brain, introduced in subsection 2.1.4. Derived from biological place cells, a network of an artificial replication of this cell type is used in [39; 43] as a representation of the environment. Thereby, the search space is divided by a 2D grid. In the neural space, each neuron is associated with one grid cell and connected to its neighbors. In contrast, for a neural implementation within the C-space, neurons represent discrete

configurations and path planning is executed on the synapses. The size of such a network depends on both the DOF of the robot and the resolution concerning the joint angles but quickly becomes extremely large. Consequently, path planning with SNN in literature is mostly applied to mobile robots [39; 367; 43; 37; 47; 49]. In [47; 48; 49] axonal delays are altered as a reaction to environmental changes. Axon conductance velocities between neurons can thus be learned to simulate costs for traversing the environment. Subsequently, neurons' spike time is recorded with Address Event Representation (AER) and the generated list of spikes is used for path planning, by searching the most recent spike of the adjacent neurons. The work of [47; 48; 49] is based on [367]. The correlated work of [367; 47; 48; 49] stands out as it was tested on neuromorphic hardware. While a custom chip is used in [367], in [48; 49] IBM's TrueNorth is used.
The methods [37; 39; 43] all use a neural lattice with similar functionality; it is topologically representative of the navigation space. Here, neighboring neurons are interconnected, and neurons representing obstacles occur in isolation. A WFA is applied on the 2D grid in all papers, thus, the activation of the neuron corresponding to the starting cell triggers a wave moving through the network. Regarding the neuron model, it can be noted that all methods use spiking neurons, but beyond that, they differ, sometimes significantly. Both methods [39; 43] are very similar as they both mimic biological place cells. Regarding two things [43] differs, firstly the use of the Hodgkin-Huxley (HH) neuron model and its robustness against noise [43]. In [43] the frequency of the target neuron, the neuron that triggers the wave, is higher than that of all other neurons. Gradually, first neighboring and then distant neurons adapt to the frequency and the network becomes increasingly synchronized. Now, when a wave triggered by the target neuron travels through the network, the frequency shift of a small area around a neuron is compared. The neural network consists of two layers, the readout layer and the planning layer. The readout layer assigns four basic movement directions to each location in the planning layer. It reads the local phase differences of the neurons in the planning layer and translates the readings into a sequence of movements in the direction of the target. To generate the correct path, these frequency shifts are used to calculate the direction back to the target neuron. The authors of [43] note that their approach is slow for large environments but can handle moving obstacles and realistic noise. In [39] synapses are strengthened through STDP in the direction of the wave. Thereby, tracking the strongest synapses yields the shortest path from the start to the goal cell. A preliminary method to this approach was presented in [38]. As well the method in [38] as in [39] are based on research describing navigation and orientation in the brain. Thereby, spatial awareness is realized with a moving cluster of neural activity and the resulting adaptations of the synapses, as described in detail in subsection 2.1.4. The used network is supposed to reproduce the hippocampus and spatial locations are mapped by cluster-like activation of the place cells. This method requires an initial orientation phase, whereby the agent moves through the entire environment while neuronal plasticity, more precisely STDP, ensures that synaptic connections of those neurons that are simultaneously active are strongly stimulated. This creates a network with synaptic connections corresponding to a map of the environment. The

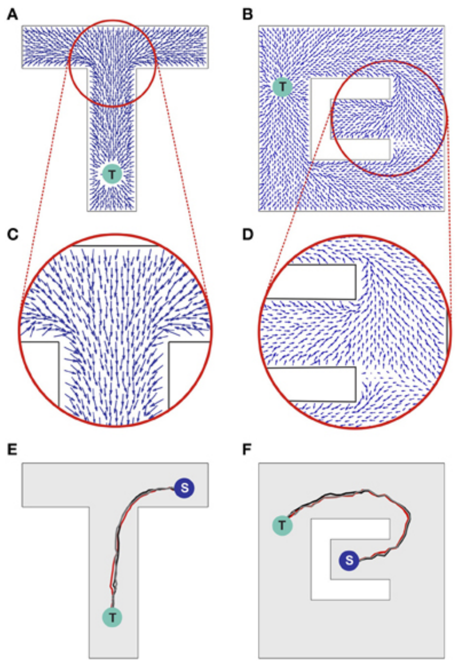
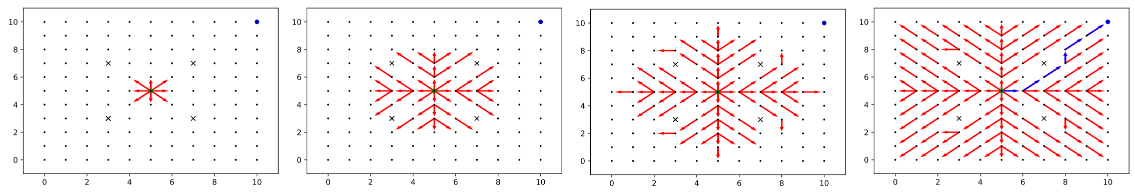


Figure 4.3.: Synaptic vector fields generated by Ponulak's neural WFA implementation [39]. The wave is initiated at the target, marked with *T*, subsequently propagating through the 2D network whereby the synapse weights are strengthened via Spike-Timing-Dependent-Plasticity (STDP), as shown in A-D. The resulting vector fields are used to generate trajectories in E and F, whereby *S* indicates the starting point. Image source: [39]

strengthening caused by STDP is interpreted as a vector that points orthogonally to the wavefront. The resulting vector field can subsequently be used for finding a path. In contrast, to [38; 39; 43], a very mathematical neural WFA with less biological relevance dispensing with neural plasticity and the concept of place cells is introduced in [37]. Thereby, each synaptic connection is given a predefined value. To initiate the wave, the place cell representing the target location is stimulated. In the process, this impulse is transmitted to synaptically connected neurons. These topological neighbors are also stimulated, thus, a chain reaction occurs and the circular pattern of a wave develops. Whether the neuron spikes has a dual dependency on the connection type to its parent neuron. Whether it is transverse or diagonal affects the differential equation for calculating a neuron's voltage as well as the threshold function for emitting a spike. Calculation of the internal voltage and the threshold still change during the simulation. Until the neuron spikes, potential parent neurons can still change. After spiking, the neuron receives an increased threshold value, so that it cannot spike again during the entire further simulation; this is not a refractory period and is actually against biological premise [44]. However, the approach [37] also implements a refractory period to prevent destructive chain reactions, such as the extinction of the initial impulse by a backtracking wave or periodic mutual excitation of two neighboring neurons. The applied neuron type is referred to as Modified Pulse-Coupled Neural Network (MPCNN) and is somewhat similar to the well-known HH model. For path finding the wave's impulse originates in the target neuron and when the start neuron spikes the simulation terminates. Since all parent-child pairs are stored, the shortest path is obtained by running the pairs of spiking neurons and their parents from the target to the start neuron in reverse order. The retrace is like following a gradient in a kind of vector field as visualized in Figure 4.4. The voltage equation of the two methods [43; 37] differs significantly. The neu-



(a) $t = \sqrt{2}$ (b) $t = 1 + 2\sqrt{2}$ (c) $t = 5$ (d) $t = 2 + 4\sqrt{2}$

Figure 4.4.: The network introduced in [37] during a neural wave from the initiation in (a) to the termination in (d). The start neuron is at the center of the map. The target neuron as well as the path found are colored in blue while obstacles are marked by an x. The time steps passed since the wave's initiation is given by t. Image source: (Steffen et al. 2020a; Liebert 2019)

rons from [43] have a randomized input and a built-in exponential current loss such that the internal current extinguishes over time without constant input. The model from [37] does not have these two components, thus, if a neuron is activated, this subsequently determines the rest of the voltage equations without any further external influence. There are no random influences, nor is there any loss of current or noise as the voltages of all neurons that were not activated rest at 0. Furthermore, the neurons in [43] fire continuously while that from [37] rest unless externally excited. The approaches also differ concerning pathfinding. In [43] the path is found throughout several wave pulses and in [37] within a single wave. Accordingly, the methodology of calculating the path is also completely different. In [43], one shimmies along the phase shift of the increased frequency, while in [37] each parent neuron is stored by each excited neuron. This parent chain is eventually walked along, provided that the signal can hit the start neuron. A common feature of [37; 43] is that the direction of the path in a circular environment of a neuron points to the neighboring neuron that fires first after it.

Planning with SNN For Robotic Arms

In [45; 46] a SNN-based method for path planning of a 6 or 7 DOF robot in an approximated C-space is presented. The approximation is achieved by coupling the task space and C-space through bidirectional feedback. To scale to the high dimensional C-space, a three-layered network model learns motions from human demonstration. The network includes, first, a fully-connected 2D net, for

Method	**Dim**	**Model**	**Neuromorph**	**STDP**	**Place cells**
[38; 39]	2D	LIF	(Loihi)	✓	✓
[37]	2D	MPCNN	✗	✗	✗
[43]	2D	HH	✗	✗	✓
[44]	2D	Izhikevitch	✗	✓	✓
[47; 48; 49]	2D	custom	TrueNorth	✗	(✓)
[45; 46]	2D +	SRM	✗	✗	✓
(Steffen et al. 2020a)	3D	MPCNN	✗	✗	✗
(Steffen et al. 2020b)	3D	LIF	SpiNNaker	✓	✓

Table 4.1.: Implementations of a neural WFA with SNN for 2D and 3D. Thereby, the neuron model (see section 2.2.1), and biological characteristics like the implementation of STDP (see subsection 2.1.1) and place cells (see subsection 2.1.4) are considered.

planning in a Cartesian x-y-plane spanning the reachable area of a robot's end-effector, which is based on [368]. As this network encodes the state of the robot the neurons are referred to as state neurons, which are connected to context neurons representing obstacles and the start and goal positions. Second, a 1D net for each DOF planning the trajectory of one joint, without sharing information during planning among each other. Third, a hierarchical network integrates the first two parts. Hence, 2D path planning is combined with the robot's kinematics. Thereby, the spikes of the 2D network are used as input to the IK models mapping motions into C-space. The authors state that the state neurons are capable of representing the surroundings as a cognitive map and thus function as a simplified abstraction of place cells.
In the course of this thesis, extensions (Steffen et al. 2020b; Steffen et al. 2020a), for motion planning of a robot arm in the task space were developed. Firstly, the neural implementation of a 3D WFA in (Steffen et al. 2020b) is based on [39] and is described in more detail in subsection 5.2.1. Secondly, the approach (Steffen et al. 2020a) is based on [37]. In (Steffen et al. 2020a) an SNN is applied, which equals a topologically organized map of the task space allowing execution of the WFA in 3D space. The approach was evaluated for 2D and 3D maps in simulation. Even though the basic functionality of this algorithm could be demonstrated, for 3D maps, there are problems regarding performance. Furthermore, as planning is done in the task space, the method suffers from redundancies and only the end effector is considered regarding collision avoidance.

4.1.3. Path Planning in a Reduced C-space

As path planning in the N-dimensional C-space is superior, due to being more efficient and direct, this alternative is preferable. The main reason why this is not chosen by default is the high dimensionality of this search space and the associated complexity of path planning. To avoid the "curse of dimensionality", the C-space can be reduced, as presented in the following. Furthermore, methods to construct a subspace that represents relevant portions of the C-space are presented. The next two sections investigate methods that approach the same problem oppositely.

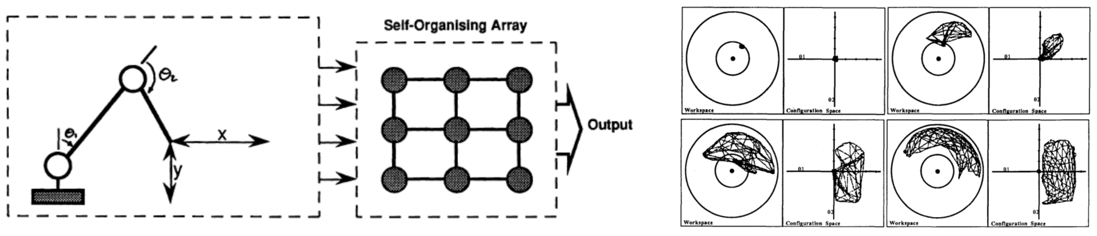
Dimensionality Reduction

The human hand is often used as a model for dimension reduction [369; 370], as it has 25 DOF, significantly more than needed for grasping and many of them are redundant. A very general, widely used approach to dimension reduction is Principal Component Analysis (PCA). In [369] it was shown that the high dimensional planning space of a human hand can be mapped to a 2D space. In mimicking this process with a robotic hand, PCA was used to generate the subspace in which the robotic hand is controlled directly. However, since this is a linear method, the relationship of the sub-space to the original C-space is also linear. In [371] a non-linear method, referred to as autoencoder, is used to reduce the C-space of anthropomorphic dual-arm robots to a low-dimensional space, the "hidden space". Therefore, the twelve joint angles of the two 6 DOF arms are used as input and transferred into the hidden features $h(x)$ by

$$h(x) = f(W_1 x + b_1). \tag{4.1}$$

Thereby b_1 is a bias vector and W_1 is the weight matrix which describes how the neurons of the original C-space are associated with the neurons of the hidden space. Path planning is subsequently performed by RRT-connect in the hidden space, in case a path is found it is mapped back into the complete C-space. The authors claim that applying RRT-connect in the hidden space results in a significant reduction of computational load and memory usage. However, the method requires a lot of training data and decoding is needed to check if generated motions are kinematically reachable and collision-free.
Scientists observed that the brain can group muscles and joints to use synergies and address these groups as one unit. The extremely high-dimensional C-space of the human body is reduced very efficiently through an unsupervised process, whereby a co-evolving structure and functional organization are established. Biologists claim that synergies are used by the brain to reduce the dimension and facilitate the determination of joint position for a specific task. Hence, the control and coordination of the human body with more than 790 muscles and 100 joints, is managed through the organization via spatiotemporal synergies which allow the use of many correlated muscles and joints as one unit [370]. A popular

method to reduce the C-space, using correlations between joint movements and thereby artificially imitating the concept of synergies between joints and muscles, are Motion Primitives (MPs) [372]. In contrast to many alternatives to overcome the curse of dimension, MPs are self-organized, and thus do not require feedback for path validation. They allow more efficient motion planning by offering a set of adjustable, "building blocks" of basic motions. Those units are represented either as splines or ANNs. They map a meaningful subset of all possible configurations and can be combined to constitute more complex motion sequences. An extension of MPs, often used in practice, is Dynamical Movement Primitivess (DMPs) [35], which can be trained by learning from demonstration. Thereby, autonomous non-linear differential equations are applied to express the motion units. As this method got a lot of attention from other scientists there are many extensions and improvements. A relevant survey is given in [373]. However, a huge drawback of this technique is that meaningful trajectories can only be achieved through excessive parameter tuning, which requires experience, is computationally intensive and time-consuming. Furthermore, learning DMPs requires a cost- or reward-function for which usually RL is applied. A huge problem is that this method also examines many meaningless movements involving cost-intensive calculations and the computational complexity increases with the number of DOF [372].
Finally, the possibility of reducing the C-space employing Self-organizing Neural



(a) (b)

Figure 4.5.: (a) Associating the position of a 2D robot's end-effector and its joint angles using a SOM. (b) Learning the position map is achieved by merging the information of the C-space and with visual data about the . Image source: (a) [374], (b) [375]

Network (SONN) is introduced. In striking contrast to the hidden feature space created by autoencoders, SONN can reduce an input space while conserving its topology. As a consequence, the topological connectivity of two adjacent configurations is guaranteed. Hence, it is possible to execute motions that were planned in the reduced subspace without the necessity of additional checks regarding the kinematic reachability or potential collisions. This advantage is given as topologically neighbored neurons in the reduced space are also adjacent neurons in the high dimensional C-space [202; 376]. Furthermore, no decoding is necessary because the SONN's neurons all contain a full set of configurations [375]. Even though there is relatively little related work in this area, a few approaches in the

'90s [375; 377; 378; 379] and '00s [380; 381] are worth mentioning. In [375] a 2D Self-Organizing Map (SOM) is applied to learn the association between C-space and task space, for a robot with 2 DOF as shown in Figure 4.5. Thereby, the system learns how a robot's joint angles are associated with its end effector position. The training data, used as input for the SOM, includes both joint angles θ_1, θ_2 and the x and y coordinates of the end-effector which is obtained through a visual component. Hence, the SOM maps 4D input onto a 2D search space, whereby each neuron represents a point in the C-space and task space. This creates a direct association between the robot's joint angle configuration and the end effector position. Thus Forward Kinematics (FK) and IK are learned implicitly without the need for any transformation equation. Path planning can be applied using a grid search in the 2D SOM and obstacle avoidance is implemented by blocking neurons with the task space coordinates of an obstacle. However, thereby only the end effector and not the whole robot arm is considered. In [383] a SOM is used

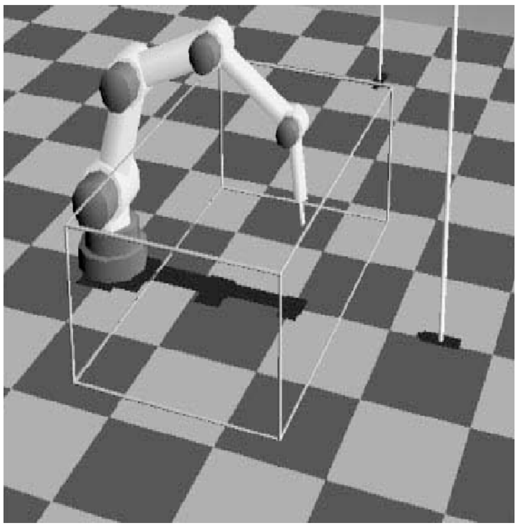


Figure 4.6.: Robotic work cell from Martinetz & Schulten, displaying the robot and vision system in simulation. A vision system with two cameras is applied and a robot with redundant DOF. The 5 DOF robot uses one joint for rotating around the vertical axis while the other four allow movements in the vertical plane. Image sources: [382]

to learn visuomotor coordination, taking advantage of the topology-preserving properties of this network architecture. Building on this, in [377] a hierarchical structure of several SOMs is used to learn positioning movements of a 3 DOF robot arm and its end effector. The visual system includes two cameras and the network architecture is a 3D SOM whereby each node is in its own a 2D SOM. As input for the 3D SOM visual 4D data is combined from both cameras and the system is supposed to form 3D joint angle configurations by using the hierarchical SOM-structure. Thereby a mapping is achieved from an object's 3D position to the joint configuration required for manipulation. However, only cylindrical objects are tested and evaluation is only done in simulation. The approach successfully maps 4D data on a 3D SOM, but it requires static visual feedback and

does not generate trajectories in the true sense, only arm positioning for a given object. Also, it does not consider dynamic but only static obstacles. Finally, [377] is not applicable for robots with redundant kinematics since exactly one joint angle node is always associated with an object position. An improved version for a 5 DOF robotic arm was published in [378]. The system, shown in Figure 4.6, proves that the extended method can also handle redundancies.
In [380] a SOM is used to plan state trajectories, which include additional constraints like velocities, accelerations or torques. The network topology used here, State Trajectory Generator (STRAGEN), is very similar to that of a Growing Neural Gas (GNG). However, the network architecture is extended by a pre-selection process of the input and a final pruning phase. This leads to good results but due to the additional complexity and the associated computing time, prevents an online application. A review about controlling a robot arm using SOMs, which does not include the newer approaches, is given in [376]. Thereby, different aspects like obstacle avoidance, hand-eye coordination and computation of IK are considered.

C-space Construction

A representation of the C-space, which allows collision-free path planning can be modeled in three ways [384]. First, as the free C-space (C_{free}) of a robot A, representing all collision-free configurations, by

$$q : A(q) \cap B = \emptyset. \tag{4.2}$$

Second, as the occupied C-space (C_{obst}), including all configurations which cause a collision, by

$$q : A(q) \cap B \neq \emptyset. \tag{4.3}$$

Or last, as the contact surface (C_{cont}), representing all configurations for which A and B are touching, by

$$\partial C_{obst}. \tag{4.4}$$

Thereby, B represents all obstacles geometrically and $A(q)$ corresponds to the robot A with the configuration q [384; 346]. C_{free} and C_{obst} jointly form the C-space and the contact surface is defined as the boundary between them. To generate these spaces, it is necessary to transfer obstacles from the task space to the C-space, which is a complicated and memory-requiring process. In the worst case, the amount of memory required increases exponentially with the DOF [384].
There are several methods of mapping an object's Cartesian to a C-space representation. Geometrical methods, try to reduce an object to geometrical features while preserving its shape. As the theoretical and implementational complexity increases exponentially with the DOF, these methods only work well for robots with 3 DOF and less [385]. In [386] the robot's C-space is constructed by use

of kinematic constraints. The authors introduce a variation on IK, which they refer to as "inverse pseudo kinematic", to analytically describe the C-space by a set of parametric equations. These equations are obtained by discretizing the task space into a grid and mapping the obstacles' boundaries to C-space using inverse pseudo kinematics. Even though this method deals better with more DOF, it struggles with an increasing number of obstacles, as the necessary checks accumulate, preventing online planning. The presented geometrical [385; 387] and analytical [386] methods construct a C_{obst}. However, the authors of [388] argue that this requires much computation time as well as memory to guarantee the completeness of C_{obst}, which is necessary for safe motion planning. Hence, in [388] the C_{free} is mapped, or more precisely areas of the C-space that are safely collision-free. By not ensuring completeness, not all collision-free configurations are captured, and computing time and memory are saved. The authors state that this method handles dynamically changing environments especially well, but, it was only tested on a 2 DOF planar manipulator. Another technique to map the C_{free} is introduced in [389], where a sample-based planer is used to build an approximation of the free C-space. The building blocks are samples generated by the RRT which have been checked regarding collisions. It is evaluated on a 6 DOF robotic platform. The method proposed in [346] uses Support Vector Machine (SVM) combined with a form of geometric approximation. Here, too, the room is sampled first, and then the samples are assigned to either C_{free} or C_{obst} through point-by-point collision checks in the task space. The algorithm based on SVMs subsequently approximates the C_{cont}, which converges quickly, also due to massive GPU-parallelization. The approach was tested on the PR2, whose arms have 7 DOF and achieve real-time capability. However, this method also does not achieve an exact representation. Lastly, bidirectional Lookup Tables (LUTs) [390; 391; 392] offer a very efficient obstacle mapping, to build up the C_{obst}. Here, both the task space and the C-space are discretized. Then, for every point $p_T \in$ task space all corresponding points $p_C \in$ C-space for which the robot would touch p_T are stored. Now, FK can be used to associate cells of the task space, representing Cartesian points with cells in the C-space, representing robot configurations. This technique is very fast, but, the memory requirements increase exponentially with the DOF. Furthermore, the memory storage is linked to the level of discretization of the bidirectional LUTs [390; 391; 392]

4.1.4. Discussion

While motion planning can also be executed in the task space, the more elegant variant is direct path planning in the robot's C-space. Hereby the robot is reduced to a point and motion planning becomes a pure path planning problem. The optimal path is deterministically provided by complete and optimal path planners like the WFA, Dijkstra's or the A* [31; 336]. In comparison, Dijkstra's and its extension the A* require more memory and calculation time than the WFA. This is due to the WFA's simplicity which can also deal well with unusual maps and

obstacle shapes [24].
However, obstacles must be transferred into the C-space and the obtained search space's dimension is related to the DOF. Thus, for this use case, complete and optimal path planners are inefficient, as they suffer strongly from the curse of dimensionality [334]. In contrast, probabilistic sample-based planners sample the C-space randomly and use connections between the samples for a graph search. In general, they are more efficient and thus better suited for high-dimensional motion planning than grid-based methods. However, due to abandoning the concept of explicitly when characterizing the C-space, they are only probabilistically complete, thus offering weaker guarantees than grid-based methods [338]. Thus, a path is non-optimal, not deterministic and not guaranteed to be found. In practice, sample-based planners have problems in cluttered scenes [346] and may lead to redundant and jerky motions which require additional smoothing [334]. Despite these drawbacks, sample-based planners are the mainstream method in a high-dimensional search space due to their far superior performance [347].
In Table 4.1, a comparison of methods for SNN-based implementations of the WFA for 2D and 3D, is given. The last column should not be taken too strictly, as it is more of a theoretical reference to the concept of place cells. Also in [47; 48; 49], the strongly biologically inspired concept is more related to other mechanisms in the brain, however, the network structure resembles place cells. Even though, the presented planning algorithms for 2D and 3D in subsection 4.1.2 all use SNN, only a small proportion makes use of neuromorphic hardware. While a self-designed neuromorphic board is applied in [367], in [48; 49] IBM's TrueNorth is used. Furthermore, the work of [38; 39], although not implemented on neuromorphic hardware in the original papers, was subsequently tested for Loihi as part of the Intel Neuromorphic Research Community (INRC) [393]. Furthermore, the method presented in (Steffen et al. 2020b) has not been implemented on neuromorphic hardware in the original paper either, however, it was realized on spiking neural network architecture (SpiNNaker) and GPU-enhanced Neuronal Networks (GeNN) as part of the performance analyses in (Steffen et al. 2021b) (see chapter 5). While most related work for SNN-based WFAs [38; 39; 37; 43; 44] is for mobile robots, thus, 2D environments, the work presented in [45; 46; Steffen et al. 2020a; Steffen et al. 2020b] differs strongly, as it is targeted to path planning for robotic arms. However, while the work presented in (Steffen et al. 2020a) & (Steffen et al. 2020b) operates in the 3D task space, the authors of [45; 46] couple the task to the C-space through bidirectional feedback. Besides the fact that [45; 46; Steffen et al. 2020a; Steffen et al. 2020b] rely heavily on the use of neuromorphic hardware to allow online application, there is another problem, regarding obstacle avoidance. In [45; 46], separate nets plan trajectories for single joints. These are combined subsequently and can only be executed in an obstacle-free map. In (Steffen et al. 2020a; Steffen et al. 2020b) path planning is performed in the Cartesian task space, hence, obstacle avoidance is not done for the robot arm but only the end effector.
Several attempts to overcome the curse of dimension by finding correlations between joint movements have been proposed, however, most techniques are not ideally suited for an online application for various reasons. For MP and DMP

several parameters need tuning, RL is necessary requiring costly explorations and most importantly, calculation efforts relate to the DOF [372]. PCA suffers from a linear relationship between the input data and reduced subspace and its extension kernel-PCA is impractical for operating on large data sets [394]. The non-linear technique autoencoder [371] offers a significant reduction of computational load and memory usage. But, the necessary decoding and collision checks, must be performed for each sample, which makes this method ineffective for online usage. Moreover, autoencoder need a huge amount of training data.
In contrast, SONNs do not require decoding of the joint angles as a full set of configurations is stored directly in each neuron. Also, SONNs can reduce a search space without disrupting its topology. Thus, neighboring neurons of the output space are topologically related in the input space. Hence, when path planning in a SONN's output space it is guaranteed that adjacent configurations are located close to each other, eliminating the need for an additional reachability check [202; 376] as necessary for autoencoder [371]. Much related work for dimension reduction of a C-space by self-organization does not exist, however, some methods have been presented. In [375] a 4D hypersurface consisting of a 2D coordinate x, y and two joint angles θ_1 & θ_2 is mapped onto a 2D space. In [377] & [378] visual 4D data from two cameras is mapped on a 3D SOM. However, these methods are mostly limited to a few DOF, 2 DOF in [375], 3 DOF in [377], 5 DOF in [378]. Furthermore, they lack an evaluation regarding real hardware. To take static and dynamic obstacles into account, a C-space construction is necessary. C_{obst} and C_{cont} require completeness for path planning, which is not the case for C_{free}. Therefore, constructing the C_{free} is computationally less costly and has lower memory requirements, which is very beneficial for joint-based motion planning. In [388] a memory-friendly and fast approach for C-space mapping is introduced, but, it does not guarantee completeness, hence, even when using an optimal path planner as the Dijkstra, it is not guaranteed that the best path is found, which makes it impractical for cluttered environments. Furthermore, [388] was only tested on a 2 DOF robot. In contrast, [389] tested on a 6 DOF and [346] on a 7 DOF robot arm, are fast techniques for C_{free} construction, achieving real-time capability. However, as neither [389] nor [346] provide an exact obstacle representation they also struggle with cluttered scenes and narrow passages. Bidirectional LUTs [390; 391; 392] are very performant, but, for robots with many DOF, the memory requirements demand a sparse discretization. Fortunately, this is exactly what is given by a reduction of the search space using SONNs.
Lastly, it is noteworthy that, even though [375] supports obstacle avoidance, just like in (Steffen et al. 2020b) not the whole robot arm but only the end effector is thereby taken into account.

4.2. A Reduced C-space for Efficient Path Planning

For modern robot applications, more and more emphasis is being placed on integration into dynamic environments. While it is easy for humans to perform goal-directed motions, while considering dynamically changing surroundings, reactive and collision-free motion planning for robots is extremely challenging. The main objective of the approach presented is, therefore, to develop and implement a reactive and collision-free path planning for robot arms with multiple DOF, inspired by the principles of biological models. As stated in [334], as robots and their applications are becoming increasingly complex, high-dimensional motion planning in the C-space is necessary to allow low processing times and fast re-planning in dynamic environments. The core idea of the proposed method

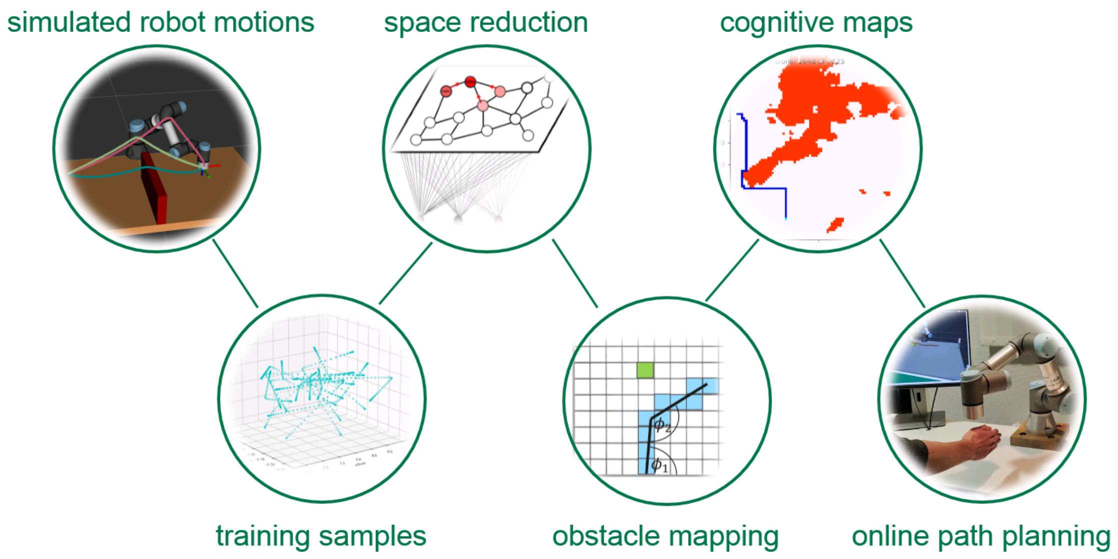


Figure 4.7.: Pipeline of the proposed neural planner which handles the high-dimensional C-space with neural self-organization. A SONN is trained by sample trajectories to represent a reduced C-space. LUT allow a fast association between cells of the and C-space, providing a real-time capable obstacle mapping that leads to an artificial cognitive map. An optimal path planner is subsequently applied. Image adapted from: (Steffen et al. 2022b)

is inspired by self-organizing brain structures, as introduced in subsection 2.1.2 and the concept of cognitive maps and place cells subsection 2.1.4. Arm movements — both artificial and human movements — are executed with high kinematic redundancies. This consideration allows, clustering of similar arm movements to obtain a lower-dimensional search space for motion planning, as introduced in subsection 4.1.3. Thereby, the complexity of the n-dimensional C-space, the search space embodying all configurations of the robot, is reduced by a SONN, taking advantage of non-linear mapping. The trained SONN's output space builds up a cognitive map, thus a neural representation of the robot's C-space. Subsequently, trajectories can be searched in this reduced space, instead of the original C-space. This makes kinematic or inverse kinematic calculations during run time unnecessary. The reduced search space allows the use of a complete and optimal path planner, a graph search algorithm, which would not be performant and practicable in the high dimensional C-space. As static and dynamic

obstacles must be transformed, a LUT is used. As analyzed in subsection 4.1.3, this allows a very efficient mapping from voxelized obstacles from the Cartesian task space to SONN's output space, the reduced C-space.
The pipeline of the approach is visualized in Figure 4.7. Initially, trajectories are generated by pick-and-place movements as training data. The trajectories are used as input to teach a SONN to represent a reduced sub-space of a robot's C-space which is covered by the training data. By use of bidirectional LUT, static as well as dynamic obstacles are subsequently mapped in the output space of the SONN, a neural C-space representation. Thus, neurons represent specific configurations, and if a configuration would cause a collision its respective neuron is blocked. The result a reduced C_{free}, can be interpreted as a cognitive map and enables the use of complete and optimal search algorithms. How SONN can be used to reduce the high dimensional C-space, preserving its topology while enabling performant path planning, is described in subsection 4.2.1. A comprehensive analysis regarding the suitability of different SONN versions, as introduced in section 2.2.2, is given in subsection 4.2.2. Lastly, how the method is extended to allow dynamical obstacle avoidance is depicted in subsection 4.2.3.

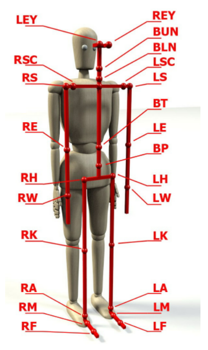
4.2.1. Reducing the Complexity of the C-space

The material covered in this section was originally published by the author in (Steffen et al. 2021c). However, in (Steffen et al. 2021c) only recorded human motions were used as training data, which is extended to robot trajectories here.
As depicted in subsection 2.2.2, SONNs can reduce a given input space's dimensionality while preserving its topological structure. The neurons of a SONN all have a weight vector $w = \langle w_1, w_2, ..., w_n \rangle$, whereby n is defined by the dimension of the input space. Generally, different input and output dimensions can be used. In the presented method for generating a reduced C-space, w corresponds to a joint configuration of the robot, thus, n is the robot's DOF. For a 6 DOF robot this lead to $w = \langle w_1, w_2, w_3, w_4, w_5 w_6 \rangle$. The SONN's output space is often 2D [179; 201]. However, mapping from a 6D input space to a 2D output space entails a great loss of information. Therefore, a higher dimensional output space is used for the presented approach. One possibility is to map arm motions to a subset of the joints, e.g. the shoulder joints or the elbow joints. The joint values of the training trajectories are given in radians. Before training, the SONN's initial weights are assigned randomly with rational values. These values are limited by the maximum and minimum joint angle values.

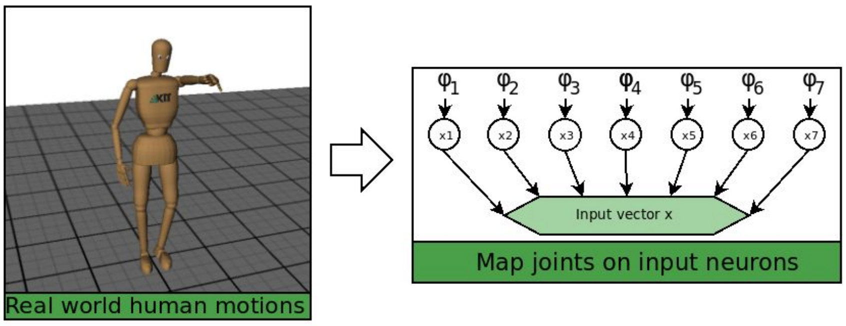
Generating Training Data

There are several ways to generate training data for the presented approach. Firstly, *(1) Recorded human motions*, thereby human arm poses from recorded motion trajectories are extracted and subsequently converted to input vectors for the network. Joint angles $\phi_1 - \phi_n$ are stored for every time step t. The Master

Motor Map (MMM) reference model [395] offers trajectories from a database of recorded human motions, as visualized in Figure 4.8(a). The data is normalized in the MMM to the height and weight of the subjects. The normalized motions



(a)



(b)

Figure 4.8.: (a) Kinematic representation of the human body by the MMM reference model, which embodies 104 DOF. (b) The seven joint angles $\phi_1 - \phi_7$ of the left arm are extracted for every time step t and used as input data for the neurons of the SONN. Image source: (a) [395], (b) adapted from (Glueck 2021)

are accessible in several formats including XML. The MMM consists of 104 DOF, thus representing the human body, however, only the left arm's seven DOF are considered in this work (LS_x, LS_y, LS_z, LE_x, LE_z, LW_x, LW_y). In Figure 4.8(b), it is visualized how the seven joints $\phi_1 - \phi_7$ of the left arm make up the input vector x.

Secondly, training data can be generated with *(2) Simulated robot trajectories*. The robot model is a UR3 controlled with the universal robotic driver package[3] and MoveIt2 [4] [396] in ROS 2 [397]. This is visualized in the 3D visualization tool for ROS (RViz) [398], in Figure 4.9. To create useful training data for general reaching motions a simple pick and place task is used. The start and end positions are randomly distributed on a surface that is parallel to the tabletop at a distance of approximately 5 cm. Additionally, the base joint's angle is constrained to ensure that the robot moves over and not around the barrier by simply rotating around the base's z-axis. For planning a probabilistic sample-based planner, the RRT-C of the Open Motion Planning Library (OMPL) [399; 400], is used. The generated data set embodies 1230 trajectories which contain 50 728 sample points.

The third possibility to generate training data is to record hand-guided robot motions, thus creating *(3) manually guided robot trajectories*. The advantage of this method is that very intuitive movements are created. Furthermore, one has a great influence on the exact nature of the trained trajectories.

What type of training data is advisable depends strongly on the intended use case. In particular, it is important to ensure that the dimensionality of the train-

[3]`https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver`
[4]`https://moveit.ros.org/`

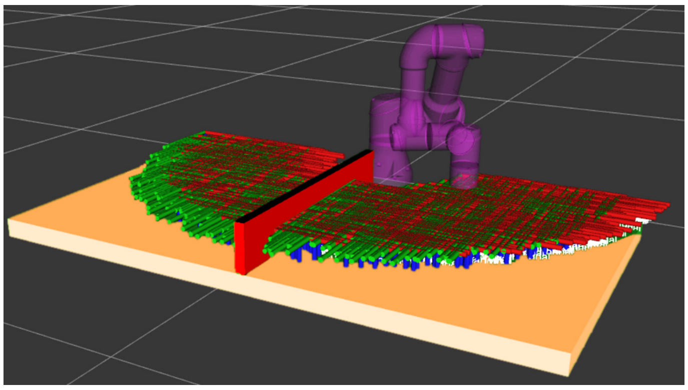


Figure 4.9.: To generate robot trajectories a UR3 is used in simulation, and motions are generated using MoveIt2. The robot is mounted on a table with a barrier. Multiple random start and goal positions are defined 5 cm above the table. The points are visualized in the picture by their coordinate systems. Image source: (Steffen et al. 2022a)

ing data is appropriate for the execution system. For training data generation according to method *(2)* and *(3)* a 6 DOF UR3e robot is used, hence the input data is 6D while the input data generated through option *(1)* is 7D. Option *(3)* is helpful to create smooth trajectories with a high level of intuition. However, to learn meaningful trajectories, a large amount of input data is necessary, which is easier achievable in simulation by method *(1)* or *(2)*.

Learning the Effectively used Subspace

In contrast to sample-based methods, which randomly sample the entire C-space, the approach learns a representation of the effectively used subspace of the robot's C-space. A second fundamental difference to sample-based planners is that the topology of the input data is implicitly learned. The trajectory is learned as a whole, thus, the course of its waypoints. This causes neurons that often occur together in a trajectory to have stronger connections. As a consequence, during path planning not only the closest points are connected, instead samples that often belonged together in the training data. This characteristic is strongly reminiscent of the cognitive maps that the brain generates during navigation, see subsection 2.1.4. The SONN's neurons represent place cells that are associated with a specific point in the 3D space and its synapses control the activation in a topologically reasonable manner.
In each cycle during learning, a configuration vector x is presented to the network. x has the same dimension as w and is drawn from one of the trajectories used as input data. The update step is then executed in each cycle, influenced by the neighborhood factor σ and the learning rate η, as outlined in subsection 2.2.2. Thereby, the Best Matching Unit (BMU) and its neighbors are pulled towards x, thus, in each step, the BMU, and to a lesser extent, its neighbors become a bit

more similar to the input vector, as visualized in Figure 2.13(a). The BMU is selected in each iteration by determining the lowest distance d_i between x and the neurons' weights d_i, by the normalized delta

$$d_i(x) = ||w_i - x|| \tag{4.5}$$

and its neighbors are chosen depending on the structure of the network model. In Figure 2.13(b) it is visualized how a gaseous network structure influences learning differently than a rigid one. In short, for a rigid network structure, the topological neighbors are matched during learning, and for a gaseous one, the neurons whose weight vectors most resemble x. For a trained network, the output space's neurons which are connected, represent neighbored samples in the original C-space. The original C-space is reduced as the neurons' weights of the SONN, are adapted by each learning cycle to eventually depict the sub-manifold of the C-space which is used by the robot in the training data. While generating a robot's reduced subspace, obstacles are neglected, thus colliding configurations are not regarded differently for now.

4.2.2. SONN-versions and their Characteristics

The material covered in this section was originally published by the author in (Steffen et al. 2022a). As presented in section 2.2.2 there is an abundance of extensions to the basic SOM. In Table 2.2 general issues for the practical application and solutions therefore are stated. However, some aspects are more relevant for the targeted use case than others. Thus, an investigation of several SONN models with respect to their applicability to reduce a robot's C-space is carried out in subsection 4.3.1. In particular, the quantization behavior and path preservation capabilities for the different SONN models are thereby considered. Which network types are used for the analysis in subsection 4.3.1 and the justification therefore, is discussed below. The theoretical basics and formulas of the two basic network types, SOM and Neural Gas (NG), are already included in section 2.2.2. However, only a rough outline of the more specialized network types evaluated in subsection 4.3.1 is given in section 2.2.2. In this section, on the other hand, the reasons for the model selection and the theoretical foundations of the selected networks are shown. The networks evaluated in subsection 4.3.1 are Merge Self-organizing Map (MSOM) [233], Growing Neural Gas (GNG) [203], Merge Growing Neural Gas (MGNG) [205], Segment Growing Neural Gas (SGNG) [206], γ Self-organizing Map (γ-SOM) [234] and γ Growing Neural Gas (γ-GNG) [235]. In Table 4.2 an overview is given of the nature of the parameters of these models. The SGNG is disregarded, as it is not a related network and a direct comparison is not possible.

SONN with temporal context

To represent trajectories in a meaningful way the consideration of preceding joint configurations is evident. However, as the basic SOM and NG have no memory structure these networks are unable to grasp a temporal context. In contrast, several SONN models regarding time sequences, as the Temporal Kohonen Map (TKM) [229], Recurrent Self-organizing Map (RSOM) [230], Recursive Self-organizing Map (RecSOM) [231], SOM for structured data (SOMSD) [232], MSOM [233], Merge Neural Gas (MNG) [401] and MGNG [205], supplement the basic networks with a merge or memory structure. These network types can store previous data in the form of preceding BMUs which makes them well-suited for time series analysis. There are multiple SONN models regarding time sequences, as summed up by the five categories given in section 2.2.2. However, mod-

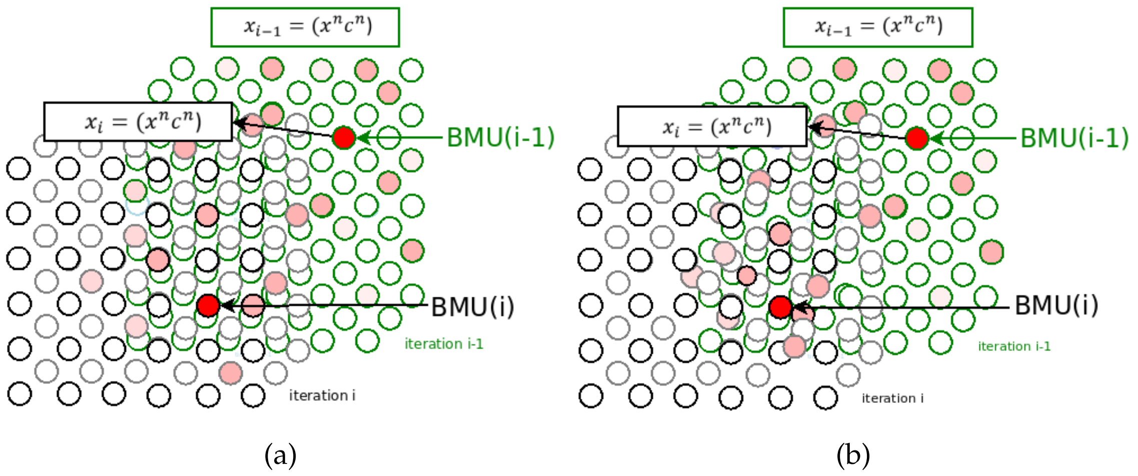


Figure 4.10.: The output space of the MNG during a learning cycle. (a) The input vector x_i embodies n joint angles x^n and a local context vector c_n, which is influenced by the BMU of $i-1$. The coloring of the neurons in red indicates which ones are affected by the update step, so the more similar a neuron's weight vector is to the BMU, the more red the corresponding neuron is. (b) The same network after the update step, the neurons in red adapted to the input and the context vector. Image adapted from: (Glueck 2021)

els which consider temporal context implicitly, like RSOM [230] and TKM [229] are considered inferior to SONN types with explicit temporal context like RecSOM [231] and MSOM [233]. The reason for this is that implicit temporal context only considers past training samples, whereas explicit context vectors integrate past learning progress. The RecSOM, while giving the most extensive representation of the temporal context, has the major disadvantage of extremely high computation times due to its highly dimensional context vector. The MSOM has an especially compact representation of explicit temporal context and is very stable. Furthermore, the weight representation is similar to the TKM and RSOM, thus, the MSOM represents a "correct" implementation of them. Due to this, RSOM, TKM and RecSOM was omitted as a candidate with temporal context and instead the MSOM and its extensions MGNG and γ-SOM and γ-GNG were investigated. For the MSOM, this changes the formulas of the SOM from section 2.2.2. In particular, the selection of the BMU given in Equation 2.7 now uses as the distance measure

$$d(x_i, a) = (1-\alpha) \cdot ||x_i - w_a||^2 + \alpha \cdot ||C_i - c_a||^2. \tag{4.6}$$

Thereby, $\alpha \in [0,1]$ is responsible for balancing present input data with the past context descriptor. C_i is a global context descriptor, a merge of the weight and context vector from the previous BMU:

$$C_i := (1-\beta) \cdot w_{BMUi-1} + \beta \cdot c_{BMUi-1} \tag{4.7}$$

The parameter $\beta \in [0,1]$ is responsible for controlling different stages of the past, thus balancing the ratio of more recent and already longer past events. It is usually set to $\beta = 0.5$. The global context descriptor is typically set $C_i = 0$ at the start of learning. Learning consists in drawing the weight vector w_{BMUi} and context vector c_{BMUi} of the current BMU and its neighbors, as well as the global context C_i, closer to the input x_i, with the learning rate η, through

$$\Delta w_a = \eta \cdot \theta(a) \cdot ||x_i - w_{BMU}|| \tag{4.8}$$

and

$$\Delta c_a = \eta \cdot \theta(a) \cdot ||C_i - c_{BMU}||. \tag{4.9}$$

The MSOM and MNG work very similarly, however, the neighborhood function $\theta(i)$ differs. For the MSOM topological neighbors are adapted as formalized in Equation 2.10 while neurons with similar weight vectors are updated for the MNG, see Equation 2.12. The learning phase is shown in Figure 4.10 exemplary for MNG. As this is a simpler neural structure, it is better suited for visualization purposes.

The γ-SOM [234] adds an adjustable memory depth, based on the γ-memory filter, to the MSOM. While each neuron i has exactly one context vector c_i for the MSOM and MNG, for the γ-SOM a set of contexts $C = \{c_1^a, ..., c_K^a\}, c_k^a \in \mathbb{R}^n$, $k = 1, ..., K$ is assigned to every neuron. The memory depth is set by K, the γ-filter order. The neuron with the smallest distance

$$d(x_i, a) = \alpha_w ||x_i - w_a||^2 + \sum_{k=1}^{k} \alpha_k ||C_k^i - c_k^a||^2 \tag{4.10}$$

in regard to the input vector x_i, is chosen as the BMU. The contribution of the weight vector is balanced by α_w and respectively the context vector by α_k, $k \in [1-K]$. K also defines the depth of the global context descriptor, which has to be calculated in each step and is defined as

$$C_k^i = \beta \cdot c_k^{BMU_{i-1}} + (1-\beta) \cdot c_{k-1}^{BMU_{i-1}}, \forall \in [1-K] \tag{4.11}$$

for the current step i. Thereby the context vector $c_k^{BMU_0}$ is initially set to 0. In order to prevent Quantization Error (QE) caused by the recursive nature of the context vectors $\alpha_w > \alpha_1 > \alpha_2 > ... > \alpha_K > 0$ is recommended. The weight and context vector of the current BMU as well as its topological neighbors, is updated by:

$$\Delta w_a = \eta \cdot \theta(a) \cdot ||x_i - w_a|| \tag{4.12}$$

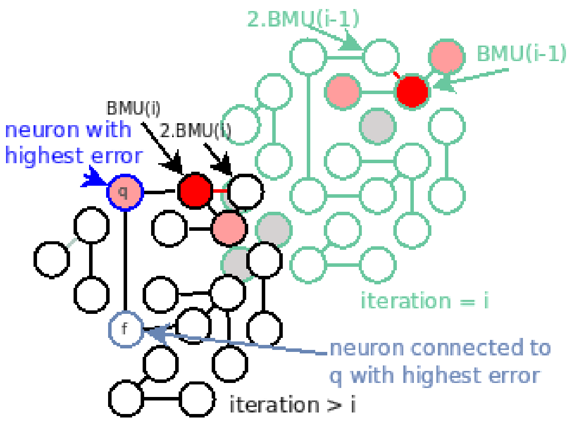
and

$$\Delta c_k^a = \eta \cdot \theta(a) \cdot ||C_k^i - c_k^a||. \tag{4.13}$$

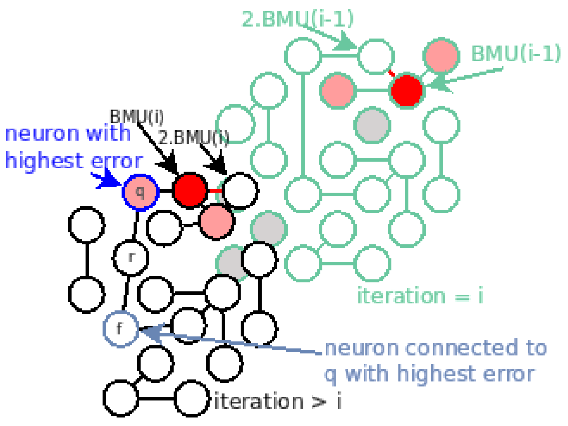
This multi-layered memory structure is visualized in Figure 4.13(a). While the neighborhood function $\theta(a)$ for the γ-SOM is Equation 2.10, the above derivation of the memory structure can be transferred to the γ-GNG, however, hereby $\theta(a)$ is given by Equation 2.12. Furthermore, it is noteworthy that the MSOM is a special case of the γ-SOM with $K = 1$, same is true for MGNG and the γ-GNG.

Growing SONN

To guarantee that the input data is represented well, the SOM as the NG require a beforehand definition of the network size. This is a big theoretical problem as it has a great influence on the QE and thus also on the topology preservation. As stated in section 2.2.2, growing networks like the GNG [203] overcome this issue by inserting neurons successively until a stop criterion is met. The learning rule


2.BMU(i-1)
BMU(i-1)
BMU(i)
neuron with highest error
2.BMU(i)
q
f
iteration = i
neuron connected to q with highest error
iteration > i

(a)


2.BMU(i-1)
BMU(i-1)
BMU(i)
neuron with highest error
2.BMU(i)
q
r
f
iteration = i
neuron connected to q with highest error
iteration > i

(b)

Figure 4.11.: (a) The GNG's output space during learning, whereby in each iteration a synapse is established between the 1st and 2nd BMU. A new neuron r is inserted between the neuron q, which has the highest error and its neighbor with the highest error f. If the edge age of a synapse exceeds a threshold it is deleted, just like unconnected nodes. (b) the output space after a learning step where a new neuron r has been inserted and neurons with a similar weight vector have been drawn to the input x. Image adapted from: (Glueck 2021)

of the GNG is closely related to the Topology Representing Network (TRN), especially regarding the simultaneous Delaunay triangulation. Hence, the GNG, like the TRN, is path preserving by obtaining optimal topology preservation. Learning differs for the GNG, in respect to the NG, in two major aspects. First, a Hebb-like learning rule within the adaptation step enables learning of topological relations with a static learning rate η. Therefore, the neighborhood rate σ of the NG is hereby obsolete. As a consequence, besides a self-adapting neuron number, the GNG has no parameters that change over time, which enables continuous learning until a performance criterion is met. Second, the network starts with two connected neurons which are extended through learning, thus the map ideally adapts to the data set. As visualized in Figure 4.11, in every learning cycle a 1st and 2nd BMU is chosen and the edge age γ is increased for all synapses which are associated with the BMU. If the 1st and 2nd BMU are not connected a synapse is established between them, otherwise for the existing synapse γ is set to 0. If a synapse's age exceeds a certain threshold it is deleted as well as all neurons which become thereby unconnected. This process ensures that the network adapts ideally to the data set as it grows neural structures where the representation is too sparse and prunes them in regions where the data is overrepresented. For the GNG a local error is added to each neuron and updated whenever the node is selected as the BMU by:

$$\Delta error(BMU) = ||w - x||^2. \tag{4.14}$$

When Λ learning cycles are completed, a neuron r is added to the population. It is inserted between q, the neuron with the highest accumulated error and f, the neighboring neuron of q with the highest accumulated error. The average of q and f weight vectors is taken as the new neuron's weight:

$$w_r = 0.5(w_q + q_f). \tag{4.15}$$

This type of neuron insertion makes the GNG highly adaptive to different sizes and dimensions of data. Furthermore, it ensures that the map grows in regions that are badly quantized, improving its overall performance. Two additional positive effects result from the GNG's structure and learning process. First, due to fixed learning parameters, the number of learning cycles is not limited. Second, incremental growth reduces the time required for learning, since not all neurons of the final network have to be completely updated in each iteration step.
The MGNG [205] combines the incremental GNG [203] with SONNs with temporal context in section 4.2.2, more precisely the MNG [401]. Thereby, a model is created that has the benefits of self-adapting map size and constant learning parameters from growing versions, as well as an explicit temporal context of merging versions. The BMU is chosen, as for the MNG, by Equation 4.6. Thus, the distance of its weight vector w_a to the current sample x_i as well as the distance of its context vector c_a to the current global context descriptor C_i are regarded. The winning neuron has the smallest merged distance, which is balanced by α. For adaptation of the BMUs, a competitive Hebbian learning approach is used. Thereby, w_a is drawn towards x_i and c_a towards C_i. This updates the synapses between the 1st and 2nd BMU. Simultaneously, connections between all other neurons are weakened, rarely used synapses are discarded and neurons without connections are

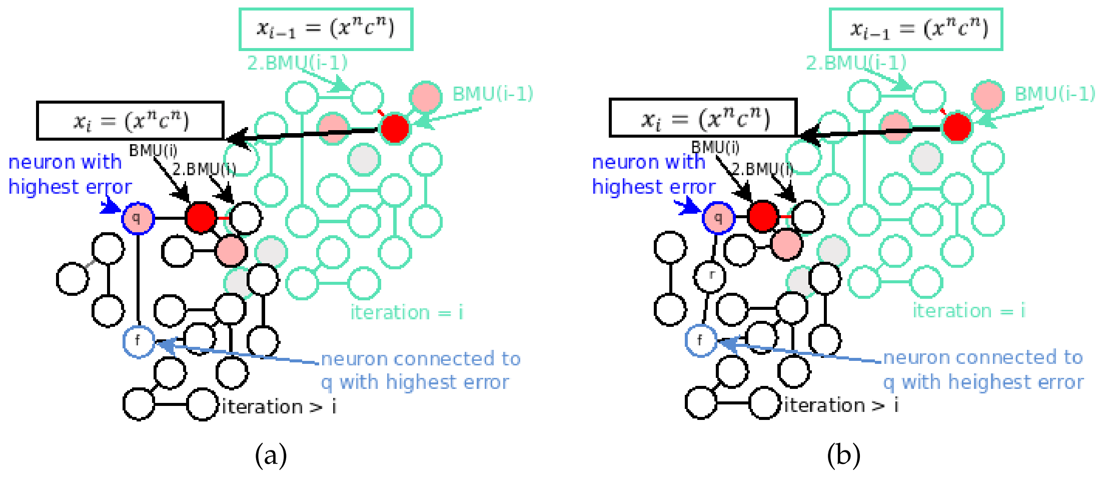

$x_{i-1} = (x^n c^n)$
2.BMU(i-1)
BMU(i-1)
$x_i = (x^n c^n)$
neuron with highest error
BMU(i)
2.BMU(i)
q
r
f
iteration = i
neuron connected to q with highest error
neuron connected to q with heighest error
iteration > i
(a)
(b)

Figure 4.12.: The output space of the MGNG during and post learning. (a) The MGNG's training vector x includes, beside the joint angles x_n, the context vector c_n. Thus, all neurons combined consist of their weight and context information. While learning, a connection is established between the 1st and 2nd BMU in each iteration. A new neuron r is inserted between the neuron q, which has the highest error and its neighbor with the highest error f. If the edge age of a synapse exceeds a threshold it is deleted, just like unconnected nodes. (b) After a learning cycle weights of the winning neurons have been drawn closer to xi and a new neuron r emerged. Image adapted from: (Glueck 2021)

deleted. This is realized, like for the GNG, with the parameter γ indicating when to delete edges and λ when to increase the map size. New neurons are inserted based on an entropy maximization regulated by Φ which differs from the GNG algorithm. For the MGNG, a map's entropy is highest in areas of a balanced activation of all neurons, thus where many neurons are stimulated frequently. This causes new neurons to be inserted in areas of frequent and evenly distributed activation. While the GNG uses an accumulation error to express the entropy, for the MGNG a counter e is introduced to each neuron, which tracks how often it was selected as the BMU. New neurons l are inserted between the most active neuron q and its most frequently activated topological neighbor f and e_q and e_f is reduced by a factor δ. Subsequently, like for the GNG, the average of w_q and w_f is used as w_l. A combination of the activation of q and f is multiplied by δ is used as the initial activation of l. Additionally, more recent changes have a stronger impact, due to decreasing all counters with the factor eta exponentially.

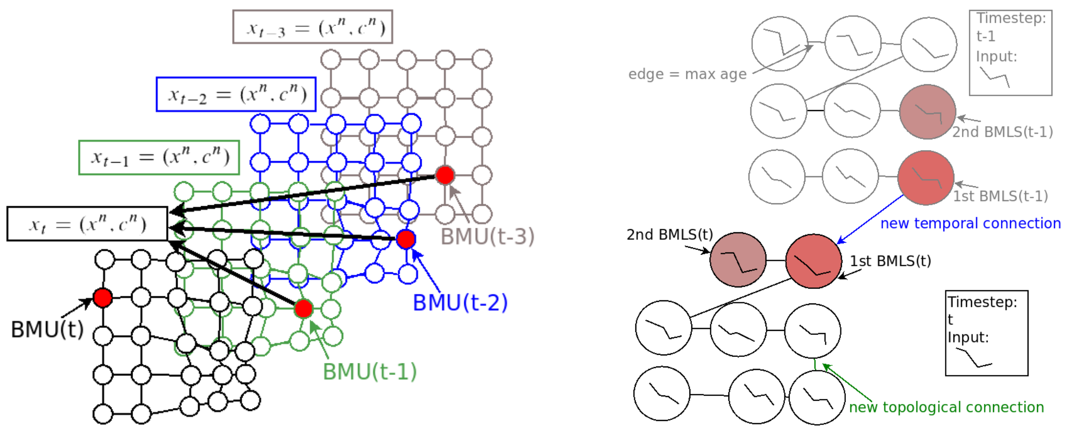
Matching linear segments

The SONN version that differs most from the other candidates is SGNG [206; 207], as it takes parts of the trajectory as input data instead of joint angles. Consequently, the basic units of quantization in the SGNG are segments, linear lines

	SOM	NG	GNG	MSOM γ-SOM	MNG	MGNG γ-GNG
η	decaying	decaying	constant	decaying	decaying	constant
σ	decaying	decaying	constant	decaying	decaying	constant
β	-	-	-	variable	variable	variable
α	-	-	-	variable	variable	variable
λ	-	-	variable	-	-	variable
γ	-	-	1 - ∞	-	-	1 - ∞

Table 4.2.: A comparison of the parameters learning rate η, neighborhood rate σ, merge factor β, merge strength α, growing step λ and edge age γ which are used in the different SONN models selected for the evaluation in subsection 4.3.1. Adapted from: (Glueck 2021)

between two neurons, instead of the neurons themselves. Thus, the focus of the



(a) γ-SOM (b) SGNG

Figure 4.13.: γ-SOM and SGNG. (a) The depth of the γ-memory is 4, thus the three previous steps are stored as temporal context. (b) The learning phase of a SGNG. A temporal connection is established between the BMLS chosen at t and $t-1$ as well as a topological one between the 1st and 2nd BMLS. Image source: (Steffen et al. 2022a)

adjustment step is no longer on the weight vector. The network considers partial trajectories in the place of the joint angles meaning that part of the trajectories are approximated by segments, which makes it a very good network for learning trajectories. Training samples are defined by a trajectory portion $\varphi_i^{t-\tau}$ and the component that wins the competitive process of the learning cycle is called Best Matching Linear Segment (BMLS), which replaces the BMU. A two-step distance measure is applied to all segments S^i to determine the BMLS. It considers, first, the spatial closeness d_{close} between the input $\varphi_i^{t-\tau}$ and each segment in S^i, in respect to their center points. Secondly, $d_{parallel}$ is calculated by the cosine similarity. It provides information about how parallel $\phi_t - \tau\phi_t$ is regarding S^i. This SONN model also considers temporal context by establishing additional temporal connections between the BMLS of two consecutive learning cycles, as visualized in Figure 4.13(b). In each step, also a connection between the 1st and 2nd BMLS is created. As the network is based on the GNG, it can grow, thus its size does not have to be predefined as for SOM-based models. The SGNG is included in the study as it is said to be good in terms of adaptation as well as temporary context representation. Furthermore, it represents a strong contrast to the other models, some of which are very similar, due to its strongly divergent structure and functionality.

4.2.3. Obstacle Avoidance

The material covered in this section was originally published by the author in (Steffen et al. 2022b). To allow path planning by use of the SONN representing the reduced C-space, the robot and the obstacles have to be transformed from the Cartesian task space into the learned network. As depicted in subsection 4.1.3, LUTs allow such a mapping efficiently, even for dynamic obstacles. Usually, the enormous memory requirements of a LUT, when used for high-dimensional spaces, is a major disadvantage. But, as only a subspace of the C-space is associated with the task space, the problem is elegantly circumvented. Hence, a considerably smaller number of joint configurations needs to be associated with the robot's task space coverage. However, as the effectively used C-space is given by the SONN, the discretization approximates the set of all required joint states for path planning. In Figure 4.14 it is visualized how an LUT is used to associate a robot's task space to its reduced C-space, for a 2D example. The task space is divided into cells, and the robot pose is determined for each learned configuration, which is contained in the individual neurons of the SONN. In Figure 4.14, this is illustrated using two neurons, marked in blue and red. Afterward, all cells in the task space, which are occupied by the robot for a certain configuration, are assigned to the corresponding neuron and this association is stored in the LUT. Now, a cell of the task space is occupied by an obstacle, in this case marked green. The association stored in the LUT can be used to immediately identify the neuron whose configuration would cause a collision, here marked red. Consequently, this neuron is then blocked for path planning in the graph structure.
The kinematic computations required for generating a LUT, refer to the used robot model, in this case the Universal Robots (UR)3 with 6 DOF. Robots essentially comprise links, basically rigid bodies, which are connected by joints [1]. The kinematics of a robot model describe the relationships of the links' position as well as velocity and acceleration, whereby forces and torques are explicitly disregarded. Robotic arms, the UR3 used here and virtually any other, are an open kinematic chain. The coordinate system of the robot arm's base represents the starting point and that of the end effector the endpoint. The kinematic chain is

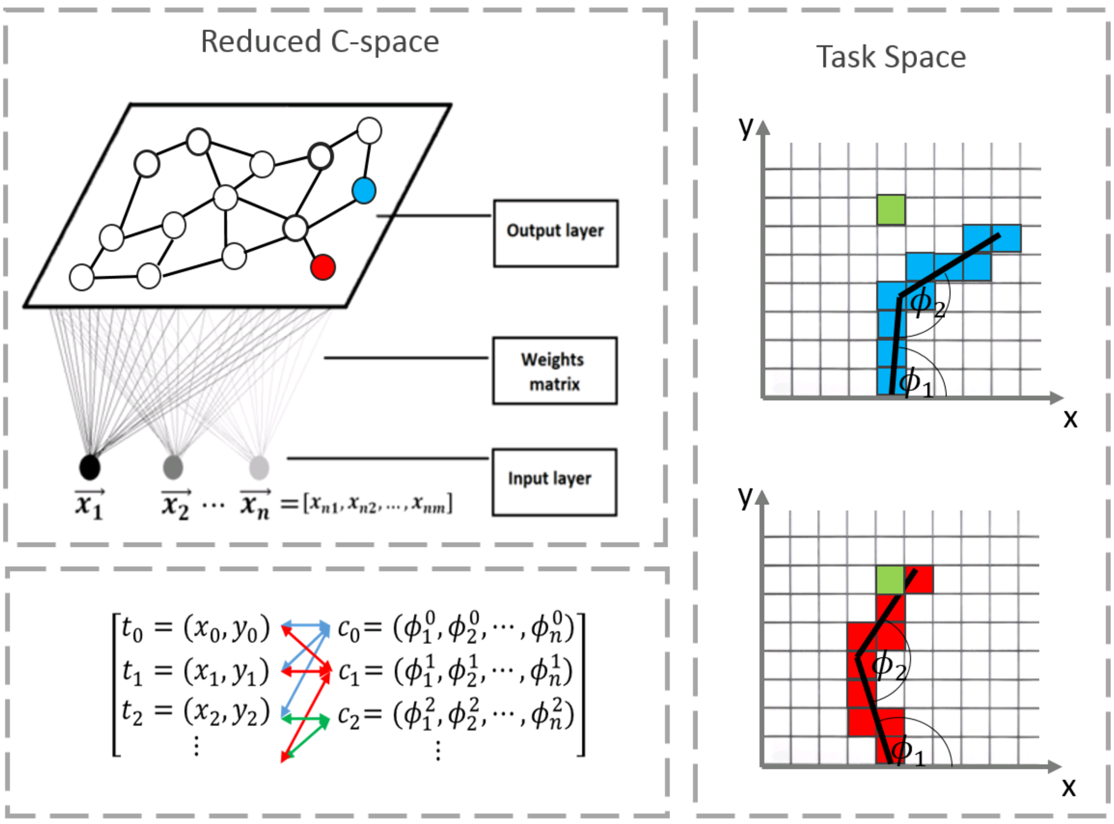

Reduced C-space
Output layer
Weights matrix
Input layer
$\vec{x_1}$ $\vec{x_2}$ ⋯ $\vec{x_n} = [x_{n1}, x_{n2}, \ldots, x_{nm}]$
$t_0 = (x_0, y_0)$ $c_0 = (\phi_1^0, \phi_2^0, \cdots, \phi_n^0)$
$t_1 = (x_1, y_1)$ $c_1 = (\phi_1^1, \phi_2^1, \cdots, \phi_n^1)$
$t_2 = (x_2, y_2)$ $c_2 = (\phi_1^2, \phi_2^2, \cdots, \phi_n^2)$
Task Space
y
x
ϕ_1
ϕ_2

Figure 4.14.: Associating the Cartesian task space with the reduced C-space by an LUT for a two-link robot. The trained SONN is visualized at the upper left, whereby each neuron represents the joint angle configuration for a specific robot pose. The association to the task space is visualized for the blue neuron in the upper right and for the red neuron in the lower right corner. The blue and red colorations in the task space mark which cells are occupied by the robot for the joint configuration $c_1 = (\phi_1, \phi_2)$. The lower left rectangle visualizes how a bidirectional LUT stores the association of the occupied task space cells t_i with the joint configurations c_i, color-coded for the respective neurons. The task space cells t_i are associated with all configurations c_i occupying the respective task space cell by the robot. In green an obstacle is marked in the task space, and the stored association allows to immediately identify the neuron potentially causing a collision. Here, the red neuron would be blocked in the graph structure to allow collision-free path planning. Image adapted from: (Weyer 2021) & [391]

represented by equations that limit the range of motion of the links. Each link has a coordinate system and their position and orientation to each other can be described in homogeneous coordinates by transformation matrices ${}_iT_k$, from the coordinate system i to k, as

$$T(\theta) = \begin{bmatrix} R(\theta) & t \\ 0 & 1 \end{bmatrix}$$

Thereby, R is the rotation matrix referring to θ in the shape 3×3, followed by the translations vector t of the form 3×1. To determine the pose of any link, the corresponding transformations are appended. For a kinematic chain with n links this would result in the kinematic equation:

$$_0T_n(\theta_1, \theta_2, ..., \theta_n) = \ _0T_1(\theta_1)_1T_2(\theta_2) ... \ _{n-1}T_n(\theta_n), \tag{4.16}$$

whereby θ_n is the joint angle vector of joint n, for the UR3 used here $n = 6$. In this context, FK is the use of Equation 4.16 to determine the end effector position from specific joint angles. The backward operation, IK, determines joint angle positions for a given pose of the end effector. In contrast to FK, IK is the significantly more computationally intensive process, also struggling with redundancies [402; 1]. In this work, however, due to the use of LUTs, only the comparatively favorable calculations of the FK are used.
The process of creating a LUT involves four steps

1. Voxelization of all single parts of the robot.
2. Individual parts of the robot are combined in a kinematic model.
3. The weight vector of all neurons, after learning, represents a joint angle configuration. The associated positions and orientations must be calculated for each link and joint by FK, see Equation 4.16.
4. Determination of all positions of the voxels representing the robot in the task space and storage of these in the LUT with the association to the corresponding neuron, representing a certain joint angle configuration.

To integrate a safety distance between the obstacles and the robot, the individual robot parts can be slightly inflated during voxelization. The bi-directional character of the LUT is important for the application. During the creation of the LUT neurons are associated with the appropriate coverage in the task space. In contrast, during run time, each cell of the task space is associated with several neurons. For each cell representing an obstacle, all neurons and thus the configurations for which they code, are determined and excluded from path planning.

4.2.4. Path Planning in a Cognitive Map

The network is trained by robot input data and after learning its nodes contain a learned joint configuration that corresponds to a specific point of the C-space. Thus, the neurons can be seen as samples of a submanifold of the C-space, in particular, the used C-space of the robot from which the training data originates. This differentiates the learned C-space representation from random sampling methods such as RRT or PRM. Instead of randomly sampled states which are spread over the whole space and need to be checked to be collision-free, the samples represented by the SONN neurons are learned in the free C-space and are only distributed in the effectively used space of the robot. Furthermore, while probabilistic planners just connect the closest samples, in the presented method, the training trajectory progression is learned implicitly by the topology of the input samples. In this sense, the trained SONN can be seen as a cognitive map (see subsection 2.1.4) learned by the robot. Thus, the output space of the SONN represents a reduced C-space of the robot.

By corresponding to certain locations in the environment, place cells allow self-localization. Additionally, as place cells can represent the current, past and future locations these brain structures allow planning the path to future positions. The newly developed method for neural path planning, presented here, transfers the biological 2D navigational concept, regarding place cells and cognitive maps, to the n-dimensional C-space of a robotic arm. Thereby, the SONN's output space forms a cognitive map and its neurons represent place cells.

Animals entering a specific location are marked by an activation pattern of place cells in the hippocampus. This is mimicked here by BMU_s referencing the initial joint configuration in the SONN. The goal configuration is given by BMU_g, and path planning is performed by a graph search algorithm. A path between BMU_s and BMU_g is found along the topological connections between the nodes. To avoid collisions with potential obstacles, neurons whose configuration would cause a collision are blocked, as outlined in subsection 4.2.3. Now, the big advantage of the method used here is that, because the C-space is not completely mapped and thus not unnecessarily inflated, an optimal algorithm (see section 4.1) can be used for path planning. Such an algorithm would be computationally too expensive for the full C-space, thus sample-based planners are usually applied. They are far more performant but not deterministic and therefore do not provide an optimal path. As the WFA, due to its parallel mode of operation, gives the prospect of rapid implementation with SNN on neuromorphic hardware, it was the first choice for an optimal path planner in (Steffen et al. 2021c; Steffen et al. 2022a). However, it has been found that a breadth-first search, like the WFA, is better suited for a grid-like search space. As the C-space corresponds more to a weighted graph than a grid, its use here is not ideal. The graph-like structure is caused by the fact that neurons have different distances to their respective neighbors because their learned weights are not evenly distributed in the C-space. As visualized in Figure 4.2, Dijkstra's algorithm considers weighted edges between nodes [403], thus its performance is evaluated against the WFA in subsection 4.3.2.

4.3. Experiments and Results

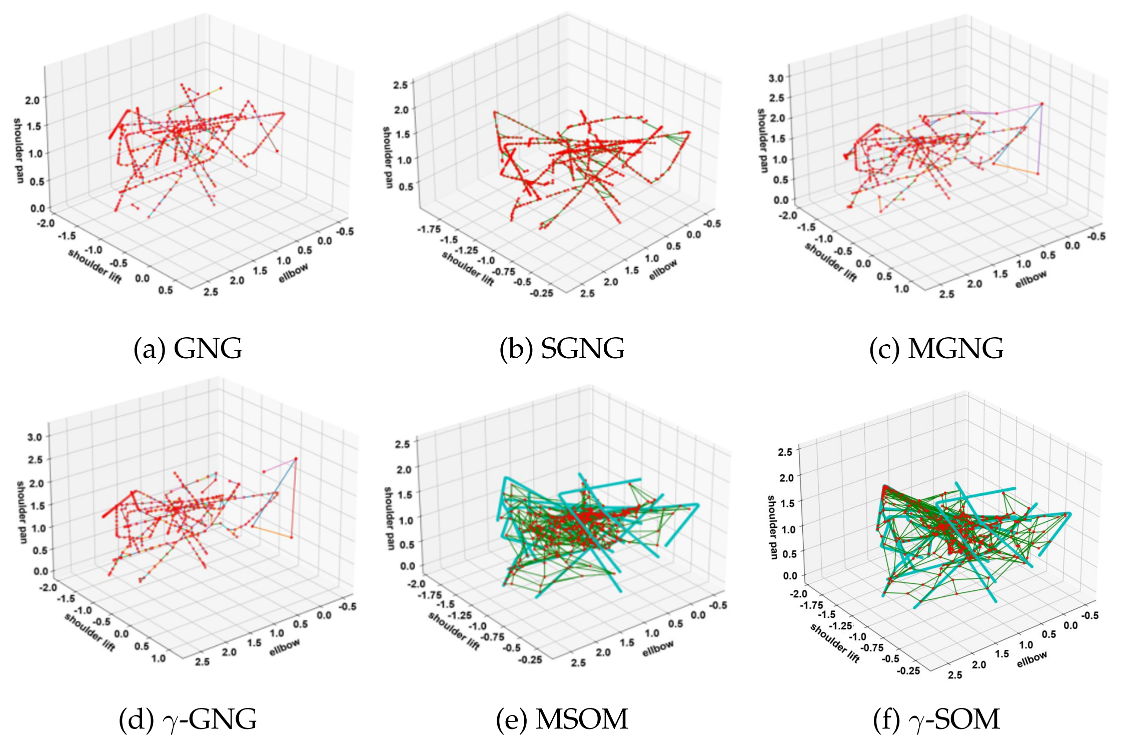
To evaluate the dimensionality reduction with the different SONN models in subsection 4.3.1, different quality measures are applied, like the QE and the C-Measure (CM). Subsequently, coverage plots as well as the generated paths are examined. This is complemented by an investigation of the suitable path-finding algorithm in the reduced C-space in subsection 4.3.2. In subsection 4.3.3 the ability to reactively avoid obstacles is tested, including a real robot demonstrator and an investigation of memory requirements. Finally, the suitability of the overall approach against popular sample-based path planning algorithms is shown in subsection 4.3.4.

4.3.1. Comparing SONN-types for Path Planning

All networks considered for the evaluation are either SOM- or NG-based versions. Preliminary tests, published in (Steffen et al. 2021c), showed that the output space of NGs and MNGs do not enable path planning, due to poor coverage of the input space. Thus they have been excluded from an in-depth investigation. Furthermore, it was established that the SOM and MSOM show very similar results. However, the MSOM was slightly superior in all aspects. Thus, the SOM was also excluded as no added value was seen in its examination. Consequently, MSOM [233], GNG [203], MGNG [205], SGNG [206], γ-SOM [234] and γ-GNG [235] are chosen for the analysis.
The study is divided into three parts. First, a qualitative visual analysis of the quantization behavior and path preservation abilities for the different SONN models. Thereby, only 15 random training trajectories are used for visualization purposes and the input comprises just three joint angles; two shoulder joints and the elbow joint. Second, a quantitative analysis of the quantization and C-space coverage of the different SONNs types, using all training trajectories. Third, an analysis of the generated 3D paths for the end effector in an obstacle-free task space.

Qualitative analysis – learning behavior and path preservation

The learned output spaces of the investigated SONN models, trained with only 15 trajectories, are visualized in Figure 4.15. Based on the literature research, it was expected that models that take the temporal context into account, would represent the input space better than the basic GNG, especially concerning path preservation. However, as shown in Figure 4.15, all GNG-based models represent the input space in a fundamentally reasonable way. However, the evaluation was quite disappointing regarding the SGNG, as it shows several splits along individual trajectories. Contrary to expectations, the basic GNG, in (a), showed similar or even better path preservation than all models with temporal context, like the



(a) GNG (b) SGNG (c) MGNG

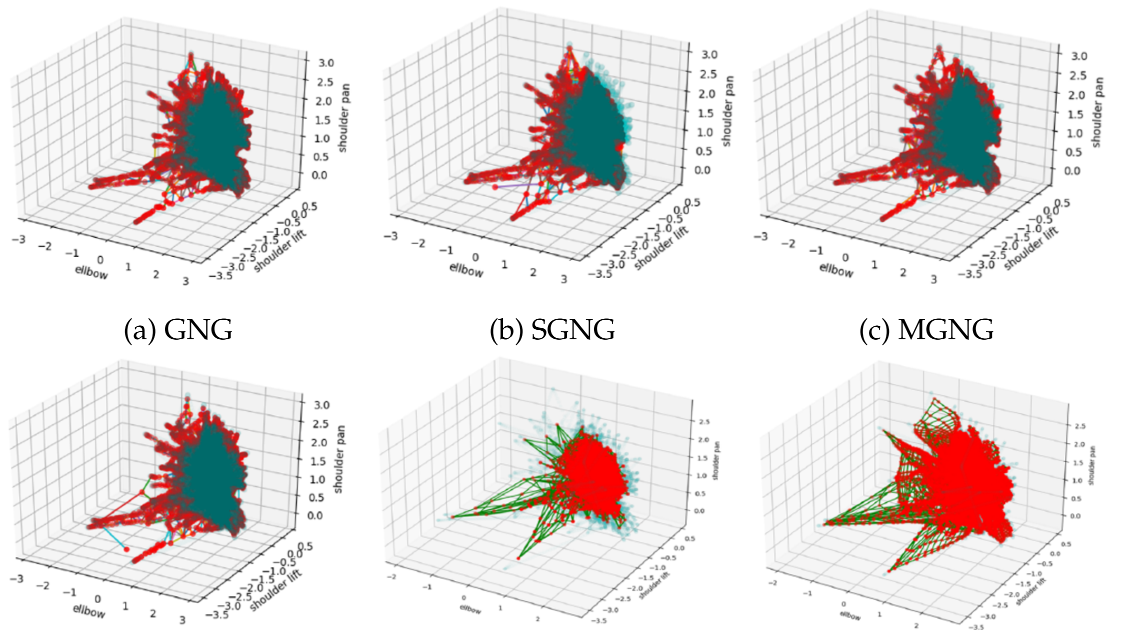
(d) γ-GNG (e) MSOM (f) γ-SOM

Figure 4.15.: Comparison of SONN models regarding learning behavior and path preservation. For this visualization, only 15 training trajectories and 3 DOF were used. The trained weight vectors are visualized by the red scatter dots and the synapses by the lines. The only exception is the SGNG in (b), green lines indicate the topological connections between the segments and the segments are visualized through red lines between two nodes. Image source: (Steffen et al. 2022a)

MGNG in (c), γ-GNG in (d) and even the SGNG in (b). Furthermore, it was observed that all GNG-based models established connections between topologically close neurons even if they originated from different trajectories. These incorrect spatio-temporal connections were expected from the GNG, but not from the more advanced models.
It is noticeable that the tested SOM-based models perform significantly worse than the GNG-based ones. This can be reasoned by two things, firstly, the high dimensional input space is not ideally mapped by low dimensional SOM structures. Secondly, SOMs are folded in space due to their topological mismatch, see Figure 2.14. This mismatch, consisting of a 2D plane that is folded to represent a 3D space, leads to points that are close together in 3D possibly being far apart concerning the folded plane. Hence, the input data is interpolated for the SOM-based models in Figure 4.15(e) & (f), illustrating their mismatch between input samples and learned weights as well as the wrongly established connections between some nodes. Lastly, a striking difference between the two basic models is that all SOM form longer connections than the GNG variations.

Quantitative analysis – C-space coverage and QE

The quantitative evaluation of all SONN versions makes use of the quality measures QE and CM, introduced in section 2.2.2. In Figure 4.16 a comparison of different SONN types regarding the C-space coverage and the QE is given. In



(a) GNG (b) SGNG (c) MGNG

(d) γ-GNG (e) MSOM (f) γ-SOM

Figure 4.16.: Comparison of SONN models regarding C-space coverage. The neurons' weights are visualized by red dots and the synapses by lines. The SOM-based models are connected to their topological neighbors while GNG types establish connections while learning. Image source: (Steffen et al. 2022a)

contrast to Figure 4.15, not only 15 trajectories are used, instead the evaluated networks are fully trained. Also, as all 6 DOF are regarded, the input data is 6D. The applied parameters are provided in section B.1 for a more in-depth discussion see (Steffen et al. 2021c; Steffen et al. 2022a).
An observation, which also agrees with the results from section 4.3.1, is that GNGs distribute better in space. Respectively, as a result of their rigid 2D structure, SOM-based network types are not capable of mapping trajectories topologically in an ideal manner. The statistical results from Table 4.3 also speak against the SOM models as they have a relatively bad QE. Furthermore, for both SOM models, many nodes are located between input trajectories without clearly identifiable affiliation. As a consequence, wide distances are linked by the synapses. Also, it is noteworthy that the topological mismatch becomes an even bigger issue for 6D, compared to the 3D scenario in section 4.3.1. In line with the previous results, there is only very little variation between different GNG types. How-

type	#N	QE	# Con. / CM	# Red. / CM
GNG	10149	0.0436	24303 / 13.9	23310 / 15.38
MGNG	10149	0.0623	54571 / 7.25	46608 / 9.88
γ-GNG	10149	0.0458	29006 / 12.34	27902 / 13.75
SGNG	8968	0.0501	24589 / 8.973	22884 / 12.29
MSOM	10000	0.1149	19800 / 61.006	–
γ-SOM	10000	0.0575	19800 / 61.737	–

Table 4.3.: SONN comparison regarding their QE and CM. #N is the number of neurons and #Con. is the number of synapses in the fully trained network. #Red. refers to the number of synapses after a reduction is applied within the GNG-based nets. Table source: (Steffen et al. 2022a)

ever, the MGNG has the worst QE from all GNG-based models. If looking at the results from Figure 4.16 & Table 4.3 together, it becomes obvious that QE and C-space coverage are related. So nets like GNG, γ-GNG and SGNG where the QE is relatively low, are also superior in terms of coverage. Respectively, the SOM models, which are clustered around the center, also have a high QE.
A consideration of the size of the trained network and the number of connections, as stated in Table 4.3, only makes sense for GNG-based models. The size of the SOM types is set and the number of synapses is indirectly also firmly defined, as the SOM cannot add and delete connections during training. The network size for the SOM types is chosen, as their rigid structure requires a fixed number of neurons. The number of 10000 neurons was chosen to correspond to the net size of the GNG-based networks to increase the significance of the comparison. The smallest net is generated by the SGNG, which is probably because in this case many connections were deleted again during the learning process. However, there is an even greater deviation in the number of synapses of the MGNG, which forms twice as many as all other models. This type of net also has the poorest CM, which suggests a causal relationship between the CM and the amount of generated connections. However, many long connections result in large joint angle jumps which impacts the path resolution badly. Furthermore, the GNG and the γ-GNG show good topology-preserving properties and also mostly short connections which are located along the input trajectories. While the other GNG models have longer synapses and more false connections, which originate from different input trajectories.

Path analysis

To complete the analysis of the different network types, a survey of the generated 3D paths for the end effector in the task space is performed. An obstacle-free task space and identical start and target configurations are used for this purpose. Therefore, in Figure 4.17 a comparison of motions generated in a subspace produced by the GNG, MSOM and γ-SOM is given. It was deliberately decided

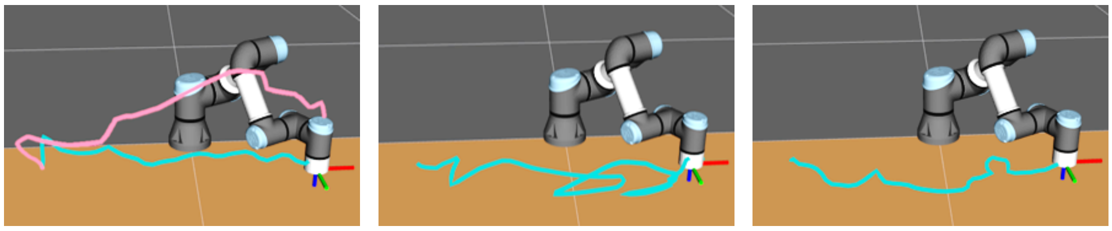

(a) GNG (b) MSOM (c) γ-SOM

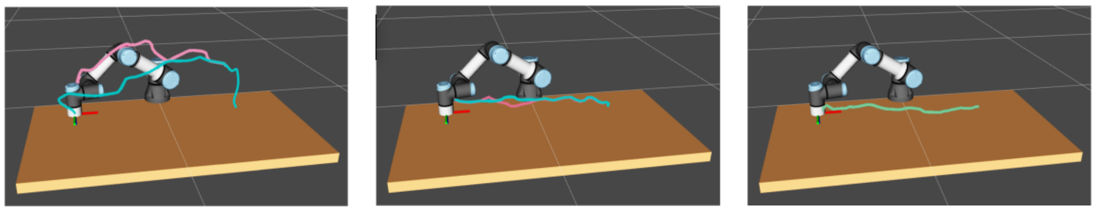
Figure 4.17.: Comparing trajectories generated in the reduced C-space by different SONN types. For all tests, identical start and target configurations are used. The path of the end effector in 3D is marked in blue. For the GNG in (a), a comparison of the networks with all connections (pink) after learning to a network with reduced connections (blue) is given. Image source: (Steffen et al. 2022a)

not to show all GNG-models as the resulting path is very similar for all networks. However, from the GNG-based networks the MGNG performs the worst, which is in line with its lower CM compared to the GNG and γ-GNG. Regarding the path generated with the SOM-types, loops and detours are particularly noticeable. A usable valid path is generated, but this is usually not an optimal path. This is caused by the topological mismatch (see Figure 2.14) between the SOM's output space in 2D and the input space in 6D. Hence, SOMs are somewhat "folded" in space and as a consequence joint states closely located in the input space are not always topologically close in the output space. However, it is noteworthy that this problem occurs less with the γ-SOM in Figure 4.17(c) than with the MSOM in (b).
For the GNG networks, an investigation was also carried out on how a subsequent reduction of the synapses using the Chebyshev distance affects path planning. This idea arose from the observation that even in GNG-based types there are sometimes paths with detours and large joint angle jumps, especially in networks with long connections. Hence, a set of reduced connections was created by use of the Chebyshev distance: $max_i|x_i - y_i|$. Thereby, long synapses linking neurons whose distance is higher than 10° are deleted. However, to prevent fragmenting the C-space, this only applies if those neurons are still connected otherwise. The longer original connections lead to a lower resolution of the generated path. The path in Figure 4.17(a) consists of 27 neurons for the original connections in pink and 32 neurons for the reduced connections in blue. This,

together with the fact that networks with original connections also create paths with more detours, means that the path quality in networks with reduced connections is significantly better. This is supported by the results of Table 4.3, showing that the CM improves for the network with reduced connections. How reducing connections affects the network quality is discussed in more depth in (Steffen et al. 2022a), also including an additional parameter study.

4.3.2. Wavefront vs. Dijkstra's

It was shown in subsection 4.3.1, that the spatial coverage of GNG-based is superior to that of SOM-based types, which leads to paths with loops and detours while GNG- based models produce nicely targeted motions. Furthermore, the basic GNG has been able to assert itself against its extensions and is therefore used for the investigation of the appropriate algorithm for path planning.
In terms of performance, the algorithms do not differ much as can be seen in Table 4.4. Even though Dijkstra's algorithm is computationally more expensive, the results show that this is negligible in practice, as the run time is less than 0.02 s in both cases, which shows the real-time capability of the approach. The main difference between the two algorithms is the way they measure distances, which has a big impact on the generated paths. WFA chooses the path with the fewest connections while Dijkstra's algorithm takes the weights of the connections into account, thus selecting the path with the lowest cost, by using the Euclidean distance. The path generation is both deterministic and reproducible, for the WFA as well as for Dijkstra's. However, a difference was noticed during evaluation, as visualized in Figure 4.18. When using Dijkstra the path remains identical if swapping the start and end points, this is not the case for the WFA. The reason



(a) Wavefront (original) (b) Wavefront (reduced) (c) Dijkstra's

Figure 4.18.: In (a) & (b) the WFA and in (c) Dijkstra is used for path planning. Additionally, in (b) the connections have been reduced before execution. Thereby, all synapses that cause joint angle jumps of more than 10° are deleted. The path for the WFA changes when the start is swapped with the endpoint, as can be seen from the two different colored paths in (a) & (b). This effect does not occur for the Dijkstra in (c). Image source: (Weyer 2021)

for this is the different distance metrics. Both methods choose always the shortest possible path, and if there are several candidates of equal merit, the first on the list is elected. The sorting of the list is deterministic but can differ if the start and end points are exchanged. As the distance measure of the Dijkstra is more accurate, the probability of a tie is significantly lower. The difference, however, which is much more serious in practice, is path quality, which is also seen in Figure 4.18. It is visible that the Dijkstra is superior to the WFA in terms of the generation of targeted and direct paths. In the SONN there are pathways with relatively long connections and few neurons that are preferred by the WFA due to its distance measure. As in chapter subsection 4.3.1 in the path analysis, a network with reduced connections was also tested in Figure 4.18(b), for comparison purposes. The path thus generated is very similar to that of the Dijkstra in Figure 4.18(c). This shows that an additional reduction of the connections significantly improves

algorithm	RC	#N	time
Wavefront	no	28	0.0124 s
Wavefront	yes	33	0.0091 s
Dijkstra	no	48	0.0131 s

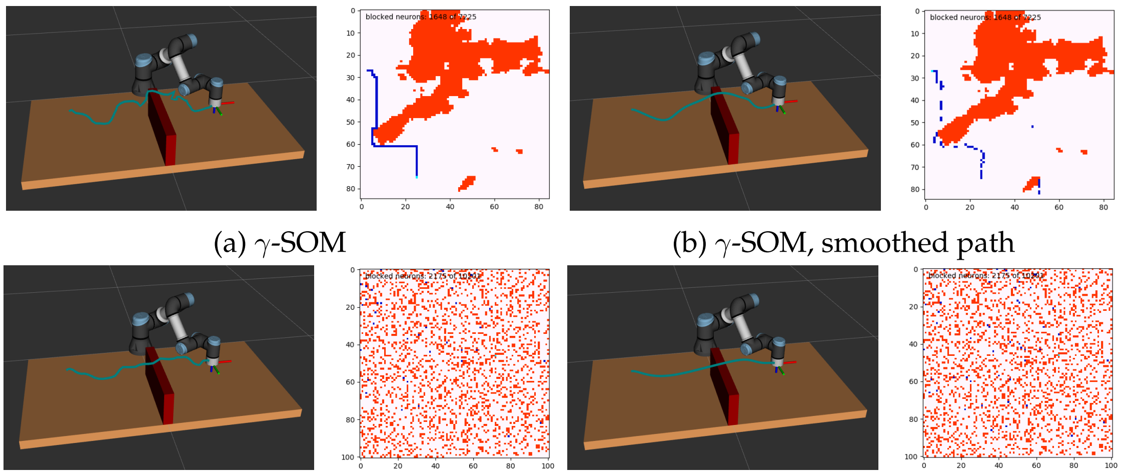
Table 4.4.: Comparison of paths generated by the WFA and Dijkstra's algorithm. For the WFA a net with the original connections is tested against one with reduced connections (RC).

the result of the WFA. However, the problem remains that due to the less precise distance measure, the best path is not chosen, which is emphasized by the deviating paths when the start and the endpoint are swapped in Figure 4.18(b). How many neurons the generated paths, shown in Figure 4.18, include, is stated in Table 4.4. Hereby more neurons are good because they mean a higher resolution. The results of the visual path analysis are confirmed as the Dijkstra has the highest path resolution followed by WFA with reduced connections.

4.3.3. Obstacle Avoidance with γ-SOM and GNG

As stated before, GNG-based can generate more targeted motions than SOM-based models, due to the folding in space, of SOMs. However, the SOM's rigid 2D structure makes it very suitable for visualizing the generated path in the reduced C-space, also referred to as the cognitive map. Furthermore, it is scientifically very interesting how this fundamentally different network type deals with obstacle avoidance, due to which the most performant SOM-type, the γ-SOM, is included in this evaluation.

In Figure 4.19 on the left side of each graphic, the end effector's path of a trajectory is shown in simulation that avoids an obstacle in the task space for the GNG and the γ-SOM. As shown in Figure 4.19(b) and (d), the generated trajectories in the task space can be significantly enhanced through additional smoothing. For smoothing non-uniform rational B-splines (NURBS) [404] are applied as the last step in the planning pipeline. This increases planning times, as stated in Table 4.5,



(a) γ-SOM (b) γ-SOM, smoothed path

(c) GNG (d) GNG, smoothed path

Figure 4.19.: Comparing the paths of an end effector in the task space and the cognitive map with and without smoothing. For the cognitive map, shown on the right side of each sub-figure, obstacles are marked in red and the path in blue. Image source: (Steffen et al. 2022b)

but only to a small extent.

In subsection 4.3.1 it was pointed out that a high number of neurons in a path

SONN	smoothed	#N	μ	σ
γ-SOM	no	76	0.007 (0.006)	0.0004
	yes		0.088 (0.085)	0.0028
GNG	no	39	0.013 (0.013)	0.0006
	yes		0.037 (0.037)	0.0012

Table 4.5.: Comparing planning times and path resolution with and without smoothing. Planning times are given with mean μ and standard deviation σ for 20 runs in seconds. The planning time of the runs that are visualized in Figure 4.19 are stated in brackets. Table adapted from: (Steffen et al. 2022b)

indicates a high path resolution and only small jumps in joint angles. This is a desirable property as sections in between two configurations are only guaranteed to be collision-free if the distances between these configurations are smaller than all possible obstacles. This is put into perspective, as the γ-SOM struggled occasionally to find a path through larger obstacles as too many neurons are blocked. The GNG performed significantly better in maps with many large obstacles. Thus, a task space that is as cluttered, as shown in Figure B.2(a) of Appendix B, is not usable with the γ-SOM but with the GNG.

Cognitive Map

In the presented method, path planning is executed in the SONN's output space. This search space also referred to as the cognitive map, is stored in a 2D array with an additional dimension for joint configurations. Using bitmaps the cognitive map can be visualized, as shown at the right side of each sub-graphic in Figure 4.19. Thereby, the bitmap's pixel represents neurons that hold joint configuration vectors. Blocked neurons, holding vectors that would cause a collision, are marked red in the image. The cognitive map of the γ-SOM is fundamentally different from that of the GNG, as can be seen when comparing Figure 4.19(a) & (b) versus Figure 4.19(c) & (d). Adjacent neurons in the visualized cognitive map are topologically connected in the output space of the SOM. Thus, obstacles are shown as contiguous areas and the path as a recognizable line. In contrast, two neighbored pixels in the cognitive map of the GNG do not represent neurons that are topologically connected, hence its output space is scattered and neither obstacles nor a path is identifiable. The reason for this difference is that the rigid 2D structure of the SOM retains neighborhoods during learning. In contrast, the topology of GNG is given by the learning of synapses. Due to this, blocked neurons of an obstacle and neurons belonging to the path do not necessarily lie next to each other in the output space. However, this is only evidence of the different structure due to the different learning algorithms, Equation 4.6 for the γ-SOM and Equation 2.13 for the GNG, and not an expression of a superior output space of the SOM. The opposite is true, as the SOM's 2D rigid structure, which allows clear visualization of the output space induces a topological mismatch with the multi-dimensional C-space. As the GNG synapses are learned explicitly they can match every topological dimensionality. So, it can be seen in Figure 4.19 that the path of the end effector in the task space is purposeful and direct for the GNG while the γ-SOM produces more twisted trajectories.
Interestingly, smoothing impairs the cognitive map as displayed in Figure 4.19(b). Hereby, gaps arise in the path and more distant neurons are added to the path. This suggests that smoothing deletes neurons that cause the strongest trajectory twists and is in line with the previous observation that path quality in the cognitive map is not related to path quality in the task space.

Demonstrator

The goal of the approach is for a robot to share a workspace with a human safely and without much downtime. To fully prove its capability, especially regarding obstacle avoidance, it was evaluated on a demonstrator which is shown in Figure 4.20.[5] This also allows the testing of realistic dynamic obstacles, such as a moving human arm. The setup is similar to the 2nd method presented for the generation of training data in subsection 4.2.1. Hence, a UR3 is operated with the UR driver package in Robot Operating System (ROS) 2 [397] which is visualized

[5]A video of the demonstrator is published at `https://youtu.be/CEkVDDg9ORw`

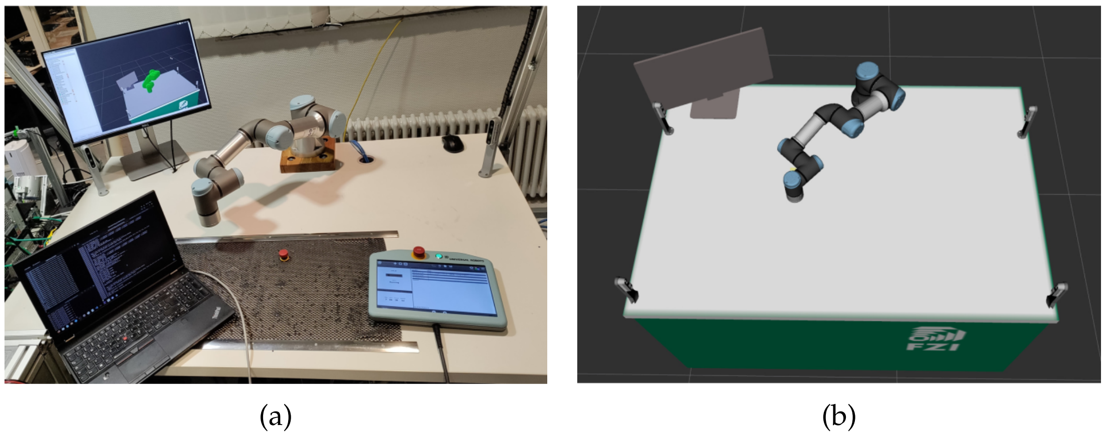


Figure 4.20.: (a) Photo and (b) visualization in RViz of the demonstrator. It includes a UR3 robot and a visual system containing four Intel RealSense.

in a RViz [398]. For the vision system four Intel RealSense D435 are mounted on each corner of a counter top and GPU-Voxels [405] is used for environment detection and voxelization. The visual processing, GPU-Voxels, is implemented with ROS1 [406] and the path planning with ROS 2, as depicted in-depth in section B.2. Therefore, the ROS 1 to ROS 2 bridge[6] was applied for communication. The visual component is executed on another PC than the planner and the ROS bridge. How a dynamic obstacle is recognized and circumvented by the demonstrator is

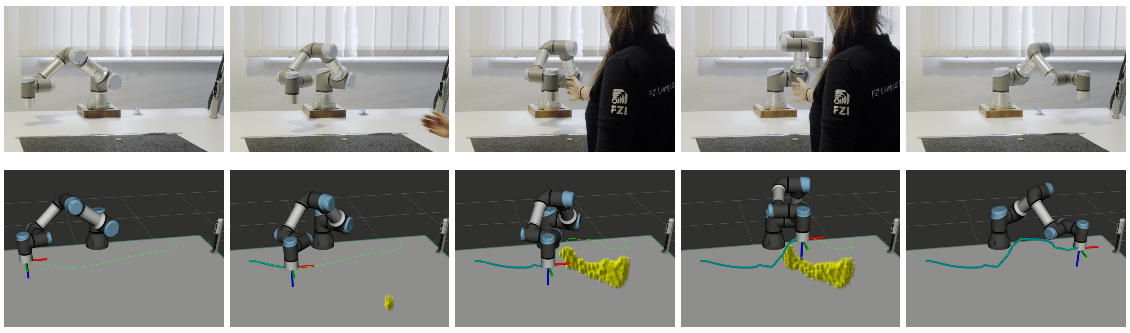


Figure 4.21.: Photo series of the reactive collision-free online motion planning. In the 1st row photos of a robot and human sharing a workspace are shown. The 2nd row displays a visualization of the scene in RViz. The human arm is captured by the visual system which is represented by yellow voxels. In green the currently planned path of the end effector is shown and in cyan the executed one.

shown step by step in Figure 4.21. Initially, a path from the start to the endpoint is planned in an empty task space. Then a hand enters the work cell and is recognized and voxelized by the vision system. The system re-plans its trajectory in less than 0.02 s and avoids the obstacle successfully.

[6]`https://index.ros.org/p/ros1_bridge/`

Memory Usage

The need for storage space in this approach is determined by the use of the LUT. The LUT has a list of neuron sets and a key vector, whose size is given by the number of cells in the task space. The key vector contains the index of a specific neuron set and is itself indexed by the cell number of the flattened task space. Therefore, the memory requirements of the LUT depend on both the size of the task space and the number of neurons in the used SONN. However, similar to the basic idea of this approach the complete C-space does not have to be learned, only the areas of the task space that are used by the training data have to be covered by the LUT. This creates an extreme difference between a theoretical worst-case scenario and an average use case in terms of memory requirements. In the scenario used for the evaluation, the task space of a UR3 was discretized to $160 \times 160 \times 160$ cells, which gives the key vector the size of 4 096 000. However, since the training data only actively uses 338 731 cells of the task space, this is also the actual number of the neuron setlist. Now, in the case of using a GNG with 10 000 nodes, 446 neurons build up a set. Hence, the theoretical memory requirement is approximately 620 MB and is made up of 4096000×4 bytes for the key vectors and $338731 \times 446 \times 4$ bytes for the neuron setlist. However, the actual memory requirement in practice was 668 MB for this particular use case, which was traced back to the overhead of the lookup class.

4.3.4. Comparisons with Modern Sample-based Planners

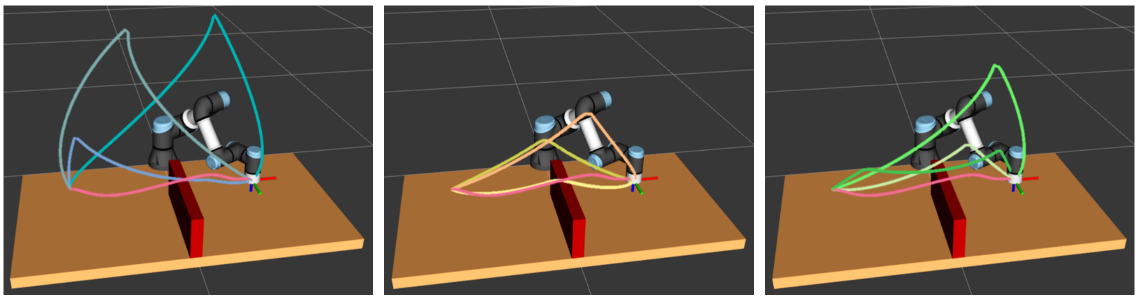
To prove the capability of the presented planner it is tested here against probabilistic sample-based planners, as these are the best performing in the field, see subsection 4.1.1. Trajectories of a Dijkstra executed in reduced C-space by a GNG are evaluated against a PRM, RRT and RRT-C. The mean value μ and the stan-

Planner	μ	σ
GNG (Dijkstra's + path smoothing)	0.037	0.0012
PRM	0.115	0.1161
RRT	2.876	1.0126
RRT-C	0.023	0.0045

Table 4.6.: Evaluation of the proposed approach against sample-based planners regarding the mean value μ and standard deviation σ. The results are for 20 runs and the planning time is in seconds. Table source: (Steffen et al. 2022b)

dard deviation σ of all tested planers are given in Table 4.6 and a visualization of the generated paths for the end effector in Figure 4.22. The sample-based planners are not generated in the reduced C-space but instead carried out MoveIt2.

To emphasize the probabilistic character of sample-based planners, several generated paths are displayed. This is not possible for the presented approach, as the Dijkstra is deterministic and provides the optimal path. These results emphasize



(a) PRM (b) RRT (c) RRT-C

Figure 4.22.: Evaluation against sample-based planners. The smoothed trajectory generated with a Dijkstra in a trained GNG, colored in red, is compared with three popular sample-based planners. As sample-based planners, in contrast to the Dijkstra, are not deterministic, several trajectories are displayed. Image source: (Steffen et al. 2022b)

the real-time capability of the proposed method and show that its planning times, including subsequent smoothing, are better than the PRM and RRT and comparable to the RRT-C. The visualization in Figure 4.22 shows that although the RRT-C is also able to generate relatively good, direct paths, none of the sample-based planners can match the approach shown in terms of path quality.

4.4. Discussion

In subsection 4.3.1 it was shown that SONN are well suited to generate a reduced C-space for path planning, as they reproduce the effectively used subspace properly. When comparing the presented approach to dimension reduction using SONNs with related work, like autoencoder [371], its advantages are readily apparent. Because the topology of the input space is maintained, kinematically reachable neighbor states are present in the output space. Also, each neuron's weight represents a full configuration state and can therefore be forwarded to the robot controller without costly post-processing. The approach presented here is comparable to sample-based planners in terms of planning time, but it can deterministically generate an optimal path, by using the Dijkstra algorithm, which would not be executable in the complete C-space. The reason for this is that only a part of the robot's task space is included in the training data and thus only a submanifold of the C-space is sampled. In contrast to probabilistic sample-based planners, these samples are not random but contain the effectively used motion space. Through self-organization, these samples are learned together with the connections, which represent the topology of the training data. Additionally, the evaluation revealed that this approach handles cluttered scenes, with many obstacles and narrow passages quite well and it is very easy to realize specific constraints regarding joint angles through the selection of training data. The approach is evaluated as well with 7D training data, generated from human motions (Steffen et al. 2021c), as also 6D training data, from a 6 DOF robot (Steffen et al. 2022a; Steffen et al. 2022b). Thereby, it was shown that the method is easily transferable to other kinematics.

Regarding the generation of training data in section 4.2.1, *(3) manually guided robot trajectories* was implemented in this work to show that it is generally feasible, but this kind of data generation is extremely time-consuming, hence it is not included into the evaluation in section 4.3. The time needed to generate a data set large enough to represent the configuration space in a meaningful way is very high. However, it can be shortened by supplementing a general set of simulated training data with hand-guided motions. When generating training data, it is noteworthy that if a part of the C-space is not included, no trajectories can be planned here. In practice, this means that if a use case is not exactly clear, it makes sense to use training data that is as general as possible. However, if the intended use case is clearly defined and uses only a small part of the task space, it is very advantageous to limit training data respectively. In this way, the memory requirements can be reduced and the quality of the generated solutions increased.

In subsection 4.2.2 different SONN versions are introduced. The MSOM [233], GNG [203], MGNG [205], SGNG [206], γ-SOM [234] and γ-GNG [235] are tested regarding their suitability for the presented approach in subsection 4.3.1. The best representatives in terms of C-space coverage are the γ-SOM and surprisingly the basic GNG, without any temporal context. Moreover, it was found that GNG-based are significantly superior regarding topology preservation than SOM-based versions, due to their topological mismatch. This problem also occurs in the path analysis, so the γ-SOM has generated paths that lead from the start to the end point without collisions, but there are often unnecessary detours and loops. However, since there are often large joint angle jumps between neighboring neurons in the GNG, generated trajectories are not always usable. The solution to successfully avoid large jumps in joint angles is deleting long connections in GNGs. Due to the structure of the GNG developing during the learning process, a trained network can be learned further by additional training trajectories. Thus, it is possible to pre-learn a GNG with general data covering the general workspace of the robot and then specify it for a new use case with task-related motions. This can reduce training time, but even more interesting, it allows to have in a network different areas specialized for different tasks, similar to the brain. This is not feasible for SOM-based models, since their size etc. must be well chosen in advance, therefore, a new network must always be trained for an additional task.

Unfortunately, there was no clear answer in the literature to the question of which network structure, is biologically more plausible. However, the assumption is that it is likely to apply to the GNG. In the NG neurons do not have real synapses with each other which is very different from the biological model introduced in subsection 2.1.1. Also SOMs do not have any explicit synapses either, but one

can imagine the synapses based on their position in the structure. In contrast, the GNG have explicit synapses that are strengthened and weakened in time, thus old connections are forgotten or refreshed. In addition, the flexible n-dimensional connection structure of the GNG seems more plausible than the rigid 2D or 3D structure of SOMs.
As discussed in subsection 4.1.2, SNN have been already used successfully for path planning in 2D. Their application to neuromorphic hardware allows asynchronous execution, increasing performance significantly in contrast to running on conventional hardware. However, the state of development as well as the availability of neuromorphic hardware currently hinders SNN-based algorithms from developing their full potential. In the course of this thesis, two methods for a neural WFA [39; 37] have been extended to a 3D environment, as published in (Steffen et al. 2020b; Steffen et al. 2020a). Apart from the fact that this is only path planning in the Cartesian task space, the implementations regarding neuro simulators on conventional hardware could not achieve satisfactory performance. Hence, an implementation on different parallel hardware architectures is done in chapter 5, to investigate the extent to which this technology affects a specific SNN implementation. For this purpose, the approach presented in (Steffen et al. 2020b) is applied, as it appears to be the more promising candidate.

5. Neuromorphic Technologies for Neural Algorithms

While analog networks are based on a differentiable activation function, Spiking Neural Network (SNN) have a membrane potential that evolves in time depending on the input of weighted spikes [6; 4]. Since these networks consider temporal dynamics they are closer to the biological model and well-suited to deal with time-related data. Instead of clean layers, they have complex structures which makes them very powerful but require complex learning algorithms. As SNN simulation requires updating the neuron state, the synapse state and the synapse's weight repetitively for each neuron, it is a highly parallel problem [161]. Their potential cannot be fully exploited when executed on a sequential von Neumann architecture [160]. Their simulation on conventional Central Processing Unit (CPU) is very expensive, however, two parallel hardware architectures exist that process SNN more efficiently, due to their ability to highly parallelize. As artificial methods are significantly less effective than their biological counterpart, neuromorphic chips, a biologically inspired dedicated hardware was developed. This technology is based on Mead's analysis of processing in the brain [162] and implements biologically plausible memory and computational elements [407]. Representatives of neuromorphic hardware are IBMs TrueNorth [172], Intel's Loihi [11], the spiking neural network architecture (SpiNNaker) system of the University of Manchester [10] and BrainScaleS developed in Heidelberg [408]. Second, Graphics Processing Unit (GPU) are also capable of parallelization [161; 160], and GPU-enhanced Neuronal Networks (GeNN) [12], a code generation library uses this feature to simulate SNN. A major benefit is that this technique is based on more accessible parallel hardware, thus it shows great potential for various applications but is still in the early stages of development. Also, neuromorphic systems are still very much in their infancy, thus, the execution of this technique is non-trivial and investigating their applicability to specific use cases is crucial, as systems may be well suited for some, but inapplicable for others [409]. Several benchmarks [59; 57; 58; 60; 62] that attempt to evaluate these differences, mainly for vision [59; 60] and from a neuro-scientific perspective [57; 58] have been introduced. However, to develop efficient SNN implementations for robotics, an application-oriented robotic scenario is necessary to investigate specification- and performance-related details of parallel systems. In [27], the Wavefront Algorithm (WFA) was suggested as a candidate as it might contribute to the greater field of neuromorphic benchmarks. This chapter addresses research question 3, defined in chapter 1: *"How can parallel hardware help to exploit the advantages of SNN?"*.

The remainder of this chapter is structured as follows. In section 3.1, an investigation of the state-of-the-art for parallel hardware is presented. Subsequently, a discussion is provided in subsection 5.1.4, which forms the basis for the following sections. In section 5.2, the core of this chapter, a performance analysis on parallel hardware is carried out for a robotic scenario. The use case for this is presented in subsection 5.2.1, followed by the technical details of the benchmark in subsection 5.2.2 and the metrics used in subsection 5.2.3. The result of the benchmark is presented in section 5.3. Derived from the presented metrics the experiments concern simulation time in subsection 5.3.1, path length in subsection 5.3.3 and finally, the consumption of hardware resources in subsection 5.3.4. The chapter is concluded in section 5.4, whereby a contextual analysis, regarding the significance of the results, is provided in subsection 5.4.1, whereupon an in-depth discussion about the limitations builds up in subsection 5.4.2.
The material covered in subsection 5.2.1 was originally published by the author in (Steffen et al. 2020b) and the rest of section 5.2 & section 5.3 was published in (Steffen et al. 2021b).

5.1. State-of-the-art

The applied systems and relevant software tools are covered in subsection 5.1.1, but the focus regarding state-of-the-art is on benchmarks for SNN simulations in literature, presented in subsection 5.1.3.

5.1.1. Parallel Hardware

As SNN simulation is a highly parallel problem [161], CPU, which process sequentially, is not the optimal choice. Nevertheless, many neural simulators run on traditional hardware, scaling from a single processor core to High Performance Computing (HPC) cluster. Thus, fast simulation of SNN can only be achieved with high energy requirements. In this section, two alternatives under the collective term, *parallel hardware* are considered. Neuromorphic hardware in general and SpiNNaker as a well-known representative, are looked at more closely, followed by a contemplation of GPU processing.

The Neuromorphic Hardware SpiNNaker

Neuromorphic chips, inspired by the brain and designed for a special purpose, differ significantly from traditional hardware, . A well-known scientific representative of a neuromorphic board is the SpiNNaker platform [10]. As it uses Advanced RISC Machines (ARM) microprocessors neuron and synapse models can be defined in a flexible manner using the software. The SpiNNaker board uses a custom-designed chip consisting of 18 ARM968 processors, connected via network on chip. All 18 processors share synchronous dynamic random-access memory (SDRAM) and each one embodies additionally a Static Random Access Memory (SRAM). The data on SDRAM is accessed less frequently. Each SpiNNaker board has a router that is connected to all ARM microprocessors, to enable spike communication. Furthermore, the router has additional interfaces for connections to the routers of six other SpiNNaker boards. Different components of a board, like a router and a processor, have individual clock speeds. Thus, it is possible to increase only the frequencies of the respective parts in case of a temporary increase in activity, which helps to reduce power consumption. However, by default, only 16 of the 18 processors run simulation code. One of the other two is spare and the last one carries out monitoring functions like maintaining the routing tables for spike communication. Even though, SpiNNaker can be applied to a wide range of problems that are solved with graphs, it is specifically designed to support large scale SNN simulation, hence, huge populations of neurons [410]. As a simple data bus would not meet the speed requirements of such a large system regarding spike transmission, an adapted version of Address Event Representation (AER) is applied.

SpiNNaker comes in two versions, a small SpiNN-3 board, the development

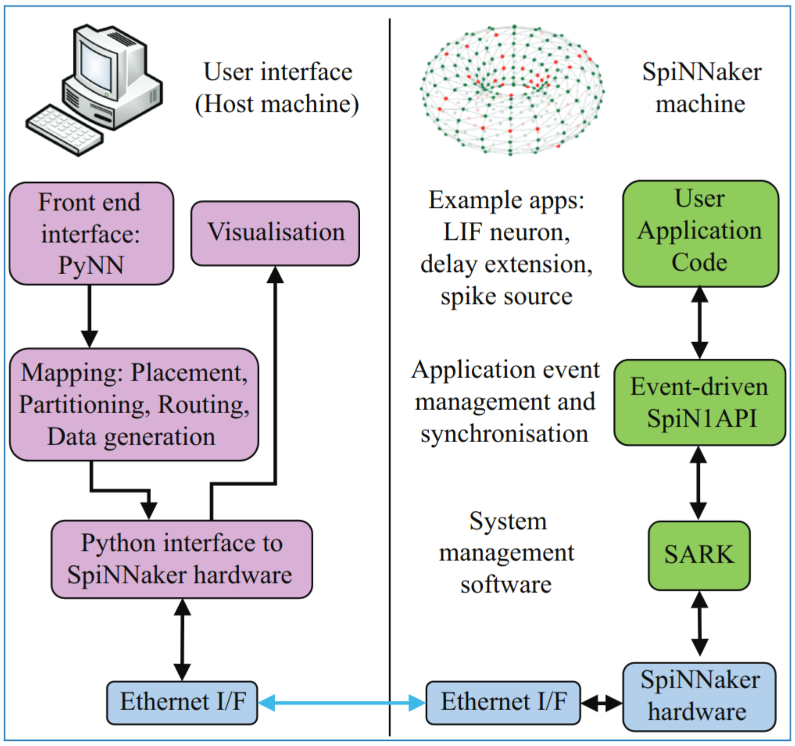

User interface (Host machine)
SpiNNaker machine
Front end interface: PyNN
Visualisation
Example apps: LIF neuron, delay extension, spike source
User Application Code
Mapping: Placement, Partitioning, Routing, Data generation
Application event management and synchronisation
Event-driven SpiN1API
Python interface to SpiNNaker hardware
System management software
SARK
Ethernet I/F
Ethernet I/F
SpiNNaker hardware

Figure 5.1.: SpiNNaker software stacks. Image source: [410]

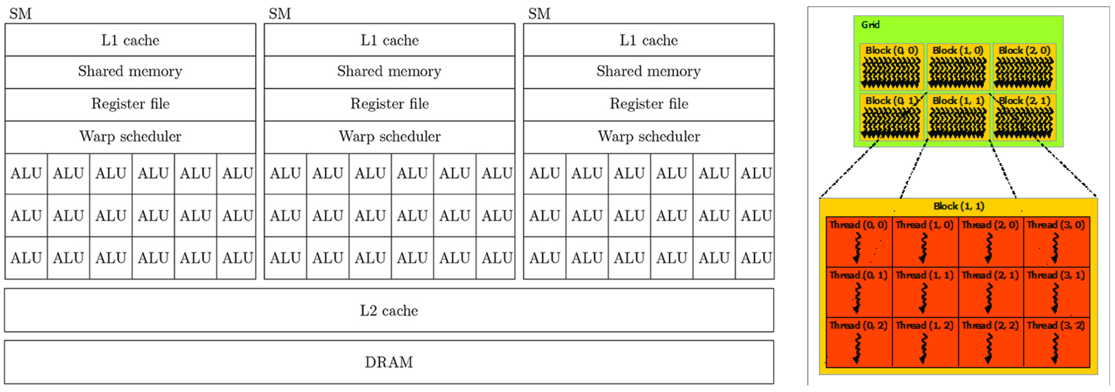
board with four chips and a larger SpiNN-5 board, the production board with 48 chips arranged in a hexagonal mesh. SpiNN-5 can be connected together and offer to attach peripheral devices, like an Event camera directly to the board. Additionally, a SpiNNaker cluster is located at the University of Manchester. The cluster, SpiNNaker1M, connects SpiNN-5 in a hexagonal mesh and embodies

1,036,800 cores.
As shown in Figure 5.1, the board is connected via Ethernet to a host machine, necessary to initialize a simulation. Figure 5.1 also visualizes that the SNN simulation on the board uses a multi-layered software stack. This modularity simplifies maintenance and as the code is open source, users can implement their code at every system level. The software stack on the host side embodies the event-based operating system SpiNNaker1 API (SpiN1API), SpiNNaker Application Runtime Kernel (SARK) enabling communication with the hardware on the chips [410]. Part of SpiN1API is SpiNNaker Control And Monitor Program (SCAMP), identifying faulty SpiNNaker cores. Before each boot-up, every core performs a self-check and the first processor successfully finishing the internal checks automatically becomes the monitor processor performing global checks on the chip. Subsequently, SCAMP instructions are loaded from the host system via the Ethernet. SCAMP is responsible to obtain a grid with all functioning cores for the simulation [411]. SpiNNTools was developed to make the loading of an application on the board user-friendly. It is used to determine which parts of the code are executed on which core, by mapping out the SpiNNaker system as a graph. This is referred to as the *MachineGraph*, whereby, cores are represented as vertices and their connections as edges. Furthermore, the time step is obtained. As the MachineGraph provides an abstraction of the hardware, it enables software to be distributed across cores. The MachineGraph is supplemented by an ApplicationGraph, whereby vertices represent applications with multiple atoms, the basic computation unit performed by a core. As application vertices are mapped onto the machine vertices software can be distributed efficiently between the cores. Both, Application- and MachineGraph, are created on the host by SpiNNTools. Users can adapt the ApplicationGraph which is subsequently converted into a MachineGraph, before being loaded on the physical board. Furthermore, SpiNNTools is provided information about the available hardware resources by SCAMP, so it can adjust the MachineGraph, if necessary to the available cores. It is also responsible for creating routing tables, defining the placement of vertices on physical cores and setting up IP tags for communication between host and SpiNNaker. After mapping, data is generated and loaded to the SDRAM of each core, before the simulation can be run. After a run, SpiNNTools generates a log file from data logged by the vertices during the simulation. The SpiNNaker board and the host PC communicate using SpiNNaker datagram packages (SDP), a protocol based on User Datagram Protocol (UDP). Also, a SpiNNaker system cannot be used standalone, it needs a host system and even though the SpiNNaker can be simulated on a pc, this is extremely inefficient.

Graphical Processing Unit

Originally developed for 3D image processing, GPUs have developed into very potent general-purpose processors for highly parallel computations in the last two decades. Parallel computations are performed on the GPU through the Single

Instruction Multiple Data (SIMD) programming paradigm [412]. Thus, as single instructions are carried out in parallel on different data, the GPU is optimized for handling high throughput. The latency is higher, especially regarding memory access, when compared with the CPU and performance is achieved by extreme parallelization. Figure 5.2(a) shows a simplified example of a modern GPU architecture. It visualizes that SIMD instructions are scheduled by Streaming Multiprocessor (SM) on the Arithmetic Logic Unit (ALU). In NVIDIA's terminology,



(a) (b)

Figure 5.2.: (a) Simplification of a GPU architecture including three SM. (b) Schematic of a CUDA grid. Image Source: (a) [58] & (b) [413]

which confusingly is manufacturer specific, an ALU is referred to as a Compute Unified Device Architecture (CUDA) core, a function is a CUDA kernel and a SIMD instruction is a warp. A warp executes only one instruction, but on up to 32 threads simultaneously, however, if not needed some might stay inactive [413]. The warp scheduler is responsible for their time-wise execution and a warp's context is stored by a register file. There are two kinds, a big one that is shared by all SM (L2) and another one as an individual cache (L1 in Figure 5.2(a)). As GPU processes are very complex and also hardware specific, NVIDIA released the CUDA API. This enables programmers to make use of the great potential of parallelized code, without requiring expert knowledge of the underlying hardware. Moreover, it allows reusing implementations on different GPUs hardware [58]. Kernels, functions in NVIDIA's terminology, are carried out in parallel by multiple threads, which are organized in thread blocks. Each block is processed by an SM, it creates threads and manages and schedules them. The amount of threads per block is constrained by the SM's capacity. However, one SM can process multiple blocks and in case two blocks are arranged identically they can process the same kernel. Blocks are grouped into 1D, 2D or 3D grids, as shown in Figure 5.2(b), and are independent, thus, can be executed in any order [413]. The CUDA memory is a hierarchical multi-layer construct. Firstly, global memory can be accessed by all threads from all blocks on all grids. Secondly, per-block shared memory is accessible by all threads from one block and thirdly, local per-thread memory. In practice it does not happen that all code is executed on the GPU, but parts on the CPU. Most architectures have individual memory for GPU and CPU, hence,

data is transferred between them. For embedded GPUs, this transfer is between the memory of the host and the device, which is quite slow. Consequently, the amount of memory transfers should be kept to a minimum. CPU memory is only accessible by 32, 64 or 128 byte transactions. Therefore, independent data transfers from neighboring units are pooled to limit the number of accesses [413].
As a consequence of the SIMD paradigm, additions on GPU are often performed separately by different threads in parallel. Due to the standard binary floating-point arithmetic (IEEE 754), which is supported by NVIDIA's GPU, floating-point numbers are non-associative. Hence, rounding errors can cause different results with additions. For example, the sum of $A + B + C$ may differ as calculations are done in parallel on different threads and $(A+B)+C$ might not equal $A+(B+C)$, due to inaccuracies caused by rounding [414]. This circumstance makes the comparison of calculations on GPU and CPU very difficult.

5.1.2. Simulation Tools for SNN

SNN simulations can be realized either synchronously, updating neuron and synapse states as well as weights at every time step, or asynchronous, updating only neurons and synapses if they received a spike. The former can be implemented through a simple matrix product, the latter, which creates greater challenges in implementation. A matrix product does not suffice here as updates do not include all state variables of the system and are not at a fixed time step [415]. Examples of synchronous SNN simulation are Brian [416] or Norse [417] which extend the machine learning framework PyTorch [418] with biologically-inspired neural components. However, many like Auryn [14], NEURON [419] and NEural Simulation Tool (NEST) [420] prefer a hybrid, where neurons are adjusted synchronously and synapses asynchronously.
An overview of tools regarding code generation for SNN simulation on neuro-

	NEST	**SpiNNaker**	**GeNN**
neuron update	synchronous	asynchronous & synchronous	synchronous
update method	exact integration	exponential integration	numeric integration
protocol	custom	AER	custom
format	-	multi cast packet	several matrices

Table 5.1.: Overview of specific mechanisms in NEST, SpiNNaker and GeNN regarding neuron, spike and synapse processing. The first two rows give information about how the systems realize neuron state update, both time-wise and regarding the method. The lower two lines refer to the transmission of the spikes.

morphic hardware, neurosimulators, as well as for execution on GPU is provided in [160]. The authors cover a variety of modeling languages and frameworks. Regarding simulators running on CPU, GENESIS [421], NEURON [419], NEST [420] and Brian [422] are introduced. Furthermore, software solutions for SNN simulation on NVIDIA GPU, like Myriad [423] and GeNN [12], and also for SpiNNaker [10] are included.
In the next paragraphs, all tools used within the benchmark are described briefly and an overview of differences regarding specific mechanisms related to the simulation of neurons, spikes and synapses is provided in Table 5.1. With regard to the timing of the neuron update, meaning how synchronous or asynchronous these individual systems are, there are certain differences. A hybrid method for simulation is used in NEST, whereby only synapses are updated asynchronously while neurons are updated at a fixed time step. In contrast, neuron updates in the SpiNNaker system are synchronous as well as asynchronous. This is realized by a individual clock for each core, while neurons updates initially start out synchronized. For GeNN, neuron and synapse dynamics are simulated continuously in the neuron kernel, thus updates are according to their model definition executed in every time step [12]. The question of the format for data transfer only arises for systems where host and device need to communicate. The asynchronous transmission protocol AER, which is also used for spike transfer in event cameras (see subsection 2.2.3), uses multi cast packets as the standard transmission format. For SpiNNaker, an adapted version of AER is employed. Using multi cast packets ensures that spikes are only sent to the chip with the target neuron, thus reducing unnecessary transmissions.

PyNN

The number of simulators and hardware solutions for SNN simulation is already high but is also constantly increasing. Many of them follow different design goals and have different strengths and weaknesses, as well as a programming language or API. From a scientific point of view, this high degree of diversity is valuable, as it allows, both, biologically more plausible and application-oriented implementations. Moreover, models can be executed and compared on different systems. However, this variety causes issues as models usually have to be translated before execution on a different system. This is a time-consuming and error-prone task, which most likely will not provide a replica, complicating evaluation. This is addressed by Python package for neuronal networks (PyNN), as models for neurons and networks can thereby be defined in a high-level language. Subsequently, PyNN models can be ported to several simulator backends or hardware solutions. Currently, supported back ends of PyNN include NEST, NEURON, Brain, BrainScaleS and SpiNNaker [424]. Moreover, a wrapper bridging PyNN with GeNN was introduced [58]. PyNN allows grouped definition as well for neurons as synapses which are of the same type. A group of neurons is called a population, whereby all neurons share their type, the postsynaptic decay, membrane time constant and resting potentials. Synapses that are defined as a group is called projections, whereby the entire construct is either excitatory or inhibitory. While PyNN allows the definition of new models, several standard types for neurons and synapses are provided. Besides modeling and execution PyNN allows the recording of parameters [424], a powerful tool for analysis purposes.

NEST

The neural simulator NEST is executable on a PC or a HPC-cluster [420]. Its native simulation language is the PostScript-based SLI and it can be interfaced through Python by PyNest [425] or with PyNN [424]. An SNN in NEST embodies two components, nodes and connections. A node is either a neuron, a population of neurons (sub-nets) or a device, thus a simulated tool for recording parameters, like the membrane potential. Connections, linking the nodes, contain optionally a weight and a delay. As weights are changeable during run time, the connections support learning rules like Spike-Timing-Dependent-Plasticity (STDP). Communication between nodes is realized through the transmission of time-stamped events, via connections. Nodes determine when to process an event by combining timestamps and delays. Several classes for events exist. The *SpikeEvent* for spike transmission based on time steps and the *PreciseSpikeEvent* which is less constrained. Furthermore, the *CurrentEvent* carries information about AC/DC and the *PotentialEvent* about membrane potential [420]. Neurons possess an ID and synapses use the ID of their post-synaptic neuron [426]. For neural processing, thus to update neuron and synapse states, exact integration [427] is used here. This has the advantage that differential equations do not have to be solved explicitly in each step. Instead the neuron state update is based on the previous step by using a propagator matrix [426].

sPyNNaker

The SpiN1API, introduced in subsection 5.1.1, is the SpiNNaker back end for PyNN. Models defined in PyNN can be transferred on a SpiNNaker board through the SpiN1API. Thus, PyNN models are translated into MachineGraph through the SpiNNTools, visualized in Figure 5.1. Thereby vertices represent populations in PyNN and edges represent the connections between them. When the simulation is completed on the board, the results are transferred back onto a host PC and retranslated by sPyNNaker code. The mapping from populations to vertices is realized so one vertex embodies the number of neurons which can be simulated by one SpiNNaker core. Neuron and synapse states are updated with exponential integration [428], meaning that incoming synaptic currents are treated as piece wise constant. The initiation of neuron updates is synchronized but they evolve asynchronously, due to independent clocks on each core. As spike processing is prioritized over neuron state updates, time drifts may occur, causing cores to get out of synchronization. The frequency of neuron updates can be specified by users, allowing them to adapt the degree of accuracy. In contrast, synapse states are only updated if a spike arrives. Spike trains are transmitted as multicast packets and stored in a spike queue, from which they are accessed through user callback functions. Hence, the ID of the sender presynaptic neuron is read from the SDRAM and then processed locally.

GeNN

The C++ library GeNN enables users to create models and generates CUDA code which runs efficiently on GPU. This opens SNN simulation on GPU without expert knowledge of low-level CUDA programming [12]. A special feature of GeNN, compared to the similar ANNarchy [429] and Myriad [423], is that users can create neuron models in C++ and are not limited to those specified as Hodgkin-Huxley (HH) or Leaky-Integrate-and-Fire (LIF). This is also the case for synapses and learning rules, whereby several common representatives are already specified. After model definition, CUDA and C++ code is generated and subsequently compiled. Then, GeNN detects the available GPU and determines a suitable CUDA block size. In case the device supports more than one GPU, GeNN performs an analysis regarding expected performance results before choosing [27]. SNN simulation is done employing three kernels, a neuron kernel, a synapse kernel and a learning kernel, which are updated at a fixed time step. To update neural states, numeric integration of the differential equation, which describes the neuron at this specific time step, is carried out. Thereby, differential equations describing the dynamics of for neurons and synapses are separated and solved individually [59]. By allowing the user to choose the exact method of numeric integration, through the exchange of code snippets, maximum flexibility is ensured [12].

5.1.3. Benchmarking Hardware and Software for SNN

Several comparisons for parallel hardware have been presented over the last few years. These benchmarks differ mainly in the selection of the systems to be tested and the benchmark scenario. An overview of the state-of-the-art regarding benchmarks is given in Table 5.2. The table includes information about the tested platforms as well as the applied use cases. The last columns states what meta language was used for implementation. The benchmarks-[59; 58; 62] chose a different path. To not be limited by a meta language that is supported by all platforms, the model was transferred to each platform.

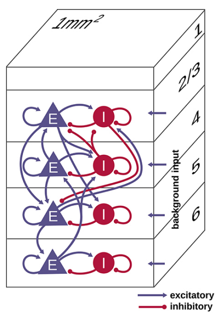
A Use Case From Neuroscience

Benchmarks focusing on a neuroscience scenario with large network simulations are given in [57; 58]. Both, [57] and [58] use the cortical microcircuit model which is introduced in [430] and visualized in Figure 5.3(a), as a test use case. It models

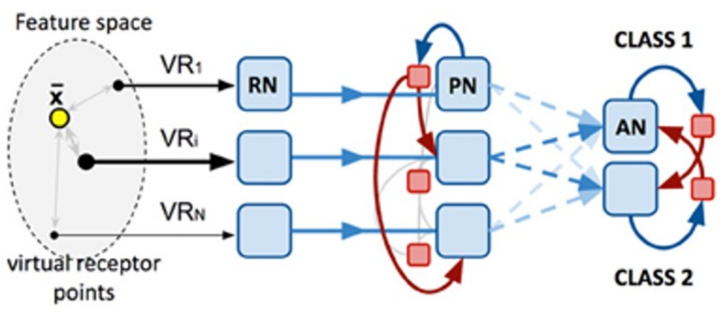
	platforms	use case	model
[59]	GeNN, Spikey, SpiNNaker	olfactory model for image classification	-
[57]	NEST, SpiNNaker	micro-column network model [430]	PyNN
[58]	GeNN, SpiNNaker, NEST	micro-column network model [430]	-
[60]	GeNN, Spikey, SpiNNaker, BrainscaleS, NEST	converted CNN for image classification	Cypress
[61]	Loihi	control of rover & force-based control of robot arm	Nengo
[62]	Loihi, SpiNNaker 2	keyword spotting & adaptive control	-

Table 5.2.: Overview of related benchmarks in literature, their target platforms and use cases. In addition, it is indicated whether a meta language was used for the implementation of the model or whether the model was implemented individually.

a slice of the cortical surface and was very likely chosen for being the smallest model that still contains a realistic number of neurons and synapses. It applies the LIF model and, due to using fixed weight synapses, learning with plasticity rules is not considered in [57; 58]. [57] evaluates the performance of NEST running on a HPC cluster and SpiNNaker running six SpiNN-5 boards. In [58], the evaluation of GeNN on different hardware is the focus while NEST and SpiNNaker are used as reference systems. Consequently, the model in [57] has been implemented in PyNN while C++ was used in [58]. Simulations in a neuroscience context, targeting a high degree of plausibility, are commonly run at a time step of 0.1 ms. The SpiNNaker supports only simulations of 1ms in real-time, it is, however, capable to decrease the simulation step to 0.1ms but this causes a slowdown by factor 20. Thus, the authors of [57] evaluate a 1 ms and an 0.1 ms time step on both systems. However, in [58] all tests are simulated with a 0.1 ms time step. The simulations regarding GeNN are carried out on several GPUs, as well as an embedded system, the Jetson TX2. Since [58] builds on [57], the same metrics were used in both papers. Simulation time and energy consumption are considered, but the focus is on accuracy, which makes sense for a neuroscience use case. In regards to the NEST simulation, the energy consumption is measured approximately once per second using the power distribution units of the HPC cluster. For SpiNNaker this is estimated by connecting a rack with 24 SpiNN-5 to a power socket, whereby only six boards are turned on and the consumed power by the rack itself is deducted. As power consumption is measured with a consumer-grade power measuring device, that does not store data at a high enough frequency, a camera is used to take a snapshot at approximately every



(a)



(b)

Figure 5.3.: SNN models used for benchmarks in literature. The network of (a) is trained for a use case in the context of neuroscience, while image classification is targeted with the olfactory model in (b). The olfactory system consists of three layers, *1* virtual receptors with associated receptor neurons, *2* projection neurons & lateral inhibitory populations, *3* a winner-takes-all-circuit of association neurons. Image source: (a) [57] & (b) [59].

simulated time step. In similar technique is used for [58] as the power draw is filmed and image recognition software is used to read out values. For determining the simulation time, both [57; 58], implement functions inside the simulation code. The tests regarding accuracy in [57], show that NEST and SpiNNaker produce very similar results, which is generalized to GeNN in [58]. For GeNN, the results regarding simulation time depend heavily on the hardware used. In the case of a high-end system, the simulation in GeNN is faster than the fastest using NEST on a HPC cluster. SpiNNaker, however, is only able to beat GeNN on the Jetson TX2.

A Use Case Regarding Image Recognition

The work in [59; 60] differs significantly from the presented papers, especially concerning the use case. The test scenario is more generally applicable, closer to a real-world application and thus, of more relevance outside the neuroscience community. Their use case focuses on image classification using the MNIST database of handwritten digits[1]. This is realized in [59] with an olfactory model, introduced in [431; 432] and visualized in Figure 5.3(b), while [60] applies a pre-trained Convolutional Neural Network (CNN). In [59] two neuromorphic platforms, Spikey and SpiNNaker, as well as the CUDA and CPU-only version of GeNN is considered. the benchmark in [60] compares three neuromorphic systems; SpiNNaker, Spikey and BrainScaleS together. It also includes NEST and GeNN on CPU and GPU. The issue that PyNN may not take all the strengths of a platform into account, is overcome in [60] by using the wrapper library Cy-

[1]`http://yann.lecun.com/exdb/mnist/`

press[2]. This C++ framework uses PyNN for NEST, Spikey and SpiNNaker while for BrainScaleS and GeNN a lower-level C++ interface is used. As [60] compares a multitude of systems with varying capabilities and scalability, it applies five different networks. Thereby, CNN are trained and subsequently converted into SNNs.
Also, for [59; 60] speed, accuracy and energy consumption are used as metrics. However, in [59] power consumption is measured only for the smallest network, thereby an external device is used and measurements are taken at 30 s intervals for 2 minutes. Regarding simulation time, the learning and testing phase are considered separately. However, in [60] the energy consumption of the BrainScaleS is estimated, as it was allegedly not feasible. According to [59] the level of accuracy achieved by GeNN and SpiNNaker is quite high, while Spikey struggles in this regard, probably due to its small network size. The surprising winner regarding simulation time in [59] is the CPU-only mode of GeNN. However, this only stands for small amounts of neurons, in large networks GeNN on GPU outperforms GeNN on CPU. Both GeNN implementations have shorter simulation times than SpiNNaker for all network sizes. The highest power consumption is recorded for GeNN on CPU, followed by GeNN on GPU and SpiNNaker, while Spikey requires the least amount of energy. The results regarding accuracy in [60] are in line with previous works, however, it is stated that analog systems achieve less accuracy than digital ones. Nevertheless, regarding energy efficiency analog systems are superior and the SpiNN-3 board and NEST occupied the last places in this category. Moreover, in [60] it is stated that GeNN on CPU is more energy efficient than GeNN on GPU for all networks. Lastly, for large networks GeNN on GPU is the fastest one, and also it outperforms SpiNNaker for all network sizes.

A Use Case Regarding Robot Control

Relatively new work related to very modern hardware is presented in [61; 62]. The work in [61] builds up on preliminary work with Nengo and embodies two robotic use cases. Firstly, control of a rover in simulation and secondly, force-based control of a robotic arm on real hardware. In [62] two use cases, keyword spotting and adaptive robotic control, are considered for evaluation. As metrics, power consumption and computation time are applied. However, the results are rather inconclusive, as the authors state that it is highly dependent on the number of input dimensions. SpiNNaker 2 performed better regarding high-dimensional vector matrices as necessary for keyword spotting while Loihi achieved better results for vector-matrix multiplication. The authors in [61] aim to provide a foundation for benchmarking robotic applications implemented in Nengo on conventional and neuromorphic architectures.

[2] `https://github.com/hbp-unibi/cypress`

5.1.4. Discussion

Robotic motion control using SNN-based implementations is an active and promising field of research. However, benchmarks are required to determine a suitable hardware solution for simulating SNN resource-friendly and performant. As system requirements are very use-case dependent, hardware comparisons, especially targeting robotics are required. The benchmarks presented in [57; 58; 59; 60], although dealing with very similar software and hardware solutions as in the presented work, apply a use case that is thematically so far that the results have no great value. Furthermore in [57; 58] learning is not considered, which is an important aspect of SNN simulation. The work in [60] embodies a plenitude of systems and applies a very elegant way to overcome the issue regarding PyNN not considering special features of some systems by using Cypress. However, as CNNs are trained and subsequently converted into SNNs, learning in SNN is not considered here either. In [58], comparing NEST, SpiNNaker and GeNN, a Jetson board is included, however, it only uses the Jetson TX2, which has been overhauled by newer boards of the Jetson series in terms of performance and thus, has no great significance.
The thematically closest to the presented benchmark of this work is [61; 62], as the use case here, is also in the research field of adaptive robotic motion control. However, [61; 62] do not include SNN simulation on GPU and focus completely on neuromorphic platforms. Furthermore, in [27], the authors propose a neural WFA, as introduced by [39], as a candidate with great impact for benchmarking parallel hardware.

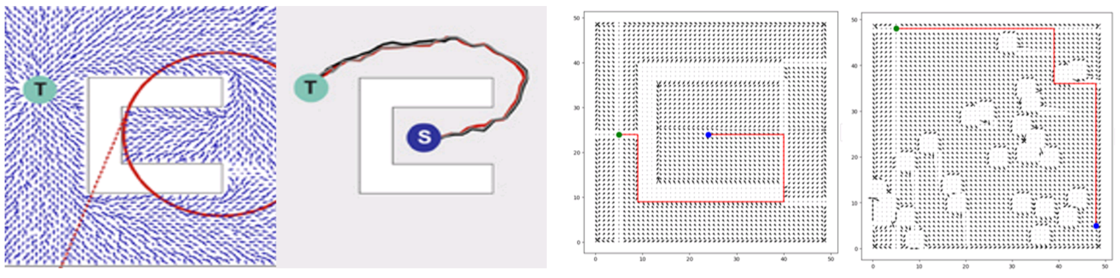
5.2. Performance Analysis of a Robotic Use Case on Parallel Hardware

As the architecture of parallel hardware solutions, differs greatly from the von Neumann architecture impacting the simulation of SNN, traditional benchmarks which are designed for conventional hardware cannot be transferred. Furthermore, the comparison of internal system metrics, such as processor clock speed, is not purposeful as it does not give any indication about performance under realistic workload [409]. Thus, the evaluation must be carried out with a realistic task. This conclusion is also reached by [59; 60]. As the implementation of the simulation environments differs so greatly it is discouraged to test only basic functions but instead analyze a suitable use case. The test scenario has been chosen to fit well with the research question 3 and still allows some generalization. Therefore, a typical robotic use case, such as path planning, is a good choice. Furthermore, it must of course be an algorithm implemented with SNN and ideally not be bound to a specific robot platform. The test scenario, hence, the algorithm to be evaluated on different parallel hardware solutions, is a neural 3D WFA (Steffen et al. 2020b) as presented in subsection 5.2.1.

The performance analysis, meaning the evaluation of the method on different software and hardware solutions, requires the test scenario from subsection 5.2.1 to be implemented in different software tools, suitable for the respective hardware solution. Related technical details are described in subsection 5.2.2. As the technical realization of the simulators differs quite strongly [59; 60] a transfer of the model is not trivial. It is desirable to influence the performance of the algorithm in the respective implementation as little as possible, to not affect the results. Consequently, a high-level modeling language is chosen, as it can be implemented on different hardware platforms with only minor adjustments. Here, the algorithm is implemented in the simulator agnostic modeling language PyNN and subsequently transferred to the respective implementations for each hardware solution. Minor adjustments remain necessary, since different neural simulators and neuromorphic hardware support different subsets of PyNN. Finally, in subsection 5.2.3, purposeful metrics for a benchmark of parallel hardware in a robotic context are defined.

5.2.1. A Neural 3D Wavefront Algorithm

The approach for a neural WFA in 3D, published in (Steffen et al. 2020b) is based on the work of [39]. As [39] is targeted at 2D environments, hence, for path planning of mobile robots, the method needs to be extended to operate in a 3D task space T. However, the dimensionality of the search space is not the only distinction between (Steffen et al. 2020b) and [39], which are shown in Figure 5.4 for a 2D map. Both methods retrieve a vector field from the synaptic weights, which have



(a) (b)

Figure 5.4.: Execution on 2D maps of the neural WFA from [39] in (a) and (Steffen et al. 2020b) in (b). Image source for (a): [39]

been learned through STDP, however, in (Steffen et al. 2020b) standard STDP and in [39] Anti-Spike-Timing-Dependent-Plasticity (Anti-STDP) is applied. For the 3D execution in a robot's task space the exploration step is skipped, as it is supposed to be provided by an external vision component. Consequently, it was also omitted in the evaluation in section 5.3. Instead, pre-existing maps have been used in simulation. Furthermore, due to the topology induced by creating an exact representation of the environment, the network's connections in (Steffen et al. 2020b) are more local than in [39]. This means that neurons have predominantly strong connections to neurons in their immediate vicinity. Also, in (Steffen et al. 2020b) LIF neurons with a fixed threshold and α-function-shaped post-synaptic current are applied while [39] uses adaptive-integrate-and-fire neurons. The most striking derivation, except the dimensionality, is the method used to determine the resulting path. In [39] the place cell representing the start is stimulated and excites its neighbors. As Anti-STDP strengthened the synapses in the direction of the goal, place cells of the optimal path are spiking significantly stronger than other neurons. In contrast, for path finding in (Steffen et al. 2020b) the agent follows the forces corresponding to the synaptic weights around the place cell which represents the agent's current position. Hence, the network's synaptic weights are interpreted as forces of a potential field. Hereby, the weight between two unoccupied neurons is stronger than between an occupied and an unoccupied one. As a result, the agent is pushed away from obstacles. The course of the WFA for

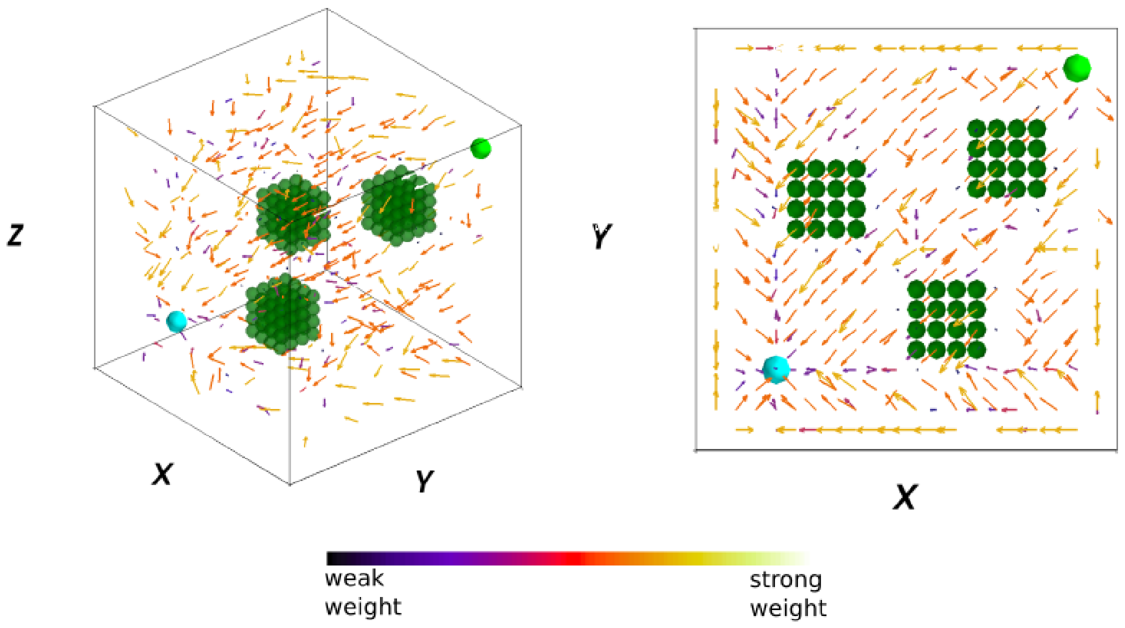


Figure 5.5.: The SVF of a map with three static obstacles and its flow, thus, the orientation and strength of its synapses, is shown on the left side in 3D and on the right side as a bird's-eye view. The view from above better illustrates the movements around the obstacles. In both graphics, the start neuron is marked cyan and the goal green. To prevent overloading the picture and allow an interpretation, a random subset of 5 % of the vectors is included and all vectors are doubled in size. The graphic shows that the vectors are oriented toward the start. Image source: (Steffen et al. 2020b)

3D presented in (Steffen et al. 2020b) can be divided into three phases, generation of a map by place cells, development of a Synaptic Vector Field (SVF) and finally the pathfinding. The SNN is a neural representation of the voxelized environment, and thereby a natural discretization of the task space. In that, each voxel is embodied by one neuron, namely, its place cell. The bidirectional synapses

connect the neurons via the Manhattan method, thus, only lateral connections are supported and synapses are not symmetrical. In this manner, each neuron is associated with the six adjacent ones. The task space is divided into free and occupied areas through the kind of applied synapses. Neurons that represent free space are connected by excitatory and neurons represent obstacles by inhibitory synapses.
To create a SVF, the WFA is used. It is initiated at the target position, by raising the membrane potential of the respective place cell, by applying an electrical current. The resulting spike stimulates the adjacent nodes, thereby initiating a neural wave that surpasses the network. As STDP is used the synaptic weights of the network are changed through the activation in the direction of the wave. Now, for each place cell n_i the subset N_{ij} is defined which contains all neurons n_j on which n_i projects directly. Furthermore, the vectors $r_i(t)$ is defined as

$$r_i(t) = \frac{\sum\limits_j w_{ij}(t)(x_i - x_j)}{\sum\limits_j w_{ij}(t)}. \tag{5.1}$$

Thereby, x_i is the vector's tip and x_j its origin and n_i's preferred position is given by x_i while x_j defines the center of gravity regarding the preferred space of its neighbors n_j. Equation 5.1 is adapted from [39], but the neuron representing the start is switched with one of the target locations, as Anti-STDP is applied in [39]. However, for visualization purposes in Figure 5.4 and Figure 5.5 the direction of the vectors' is adapted to the presentation in [39]. The visualization in Figure 5.5 shows the local direction of the wave as vectors are aligned – due to noise only almost – perpendicular to it. The vectors are additionally color-coded to allow one to easily see their strength.
Subsequently, to establish a path, the network's synapse weights are interpreted as forces locally. An agent follows the potential field from the start to the target location, thereby, the local synaptic weight vectors are averaged continuously to determine the direction of motion. In each step, the locally generated vector, which acts like a force on the agent, is added to the previous motion vector, which is then normalized before the agent moves further in the new direction. Since synapses between occupied neurons are inhibitory, the generated path naturally leads around obstacles. As the direction of movement is created locally by an average, local minima are more likely to be bypassed.

5.2.2. Technical and Implementational Details

Regarding SNN simulation, hardware and software design are closely interconnected. Thus, to select suitable candidates for the performance analysis, hardware and software solutions are considered together. Three candidates, representing strongly deviating techniques for SNN simulation, are chosen.

Simulators, Tools and Hardware

NEST is selected as the representative of an actual simulator running on conventional von Neumann hardware. The candidate for neuromorphic hardware is SpiNNaker. More specifically, the SpiNN-5 board, which can simulate significantly larger networks than the SpiNN-3 board, is applied in this benchmark. For using PyNN on SpiNNaker, sPyNNaker its PyNN interface is required [433].

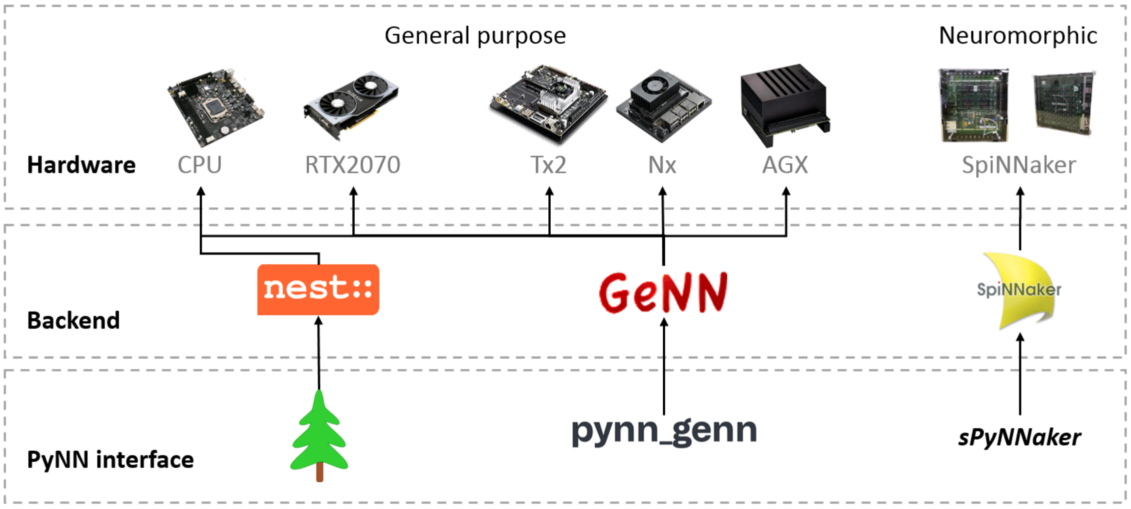


Figure 5.6.: Overview regarding the systems tested by the benchmark. The test scenario, the WFA, is modeled in PyNN. The bottom row shows Which PyNN interface is used for each backend to translate the model to the respective simulators and hardware solutions.

Lastly, to allow a comparison of neuromorphic chips to easily available hardware, GeNN is included, allowing to run SNN on GPU. For the GeNN implementation the PyNN interface for GeNN[3] is used. Nvidia's Jetson series includes several embedded GPU systems, all designed for AI applications[4]. Their small form factor enables easy integration in mobile units and the Jetson boards are therefore well suited for robotics [434]. In this benchmark, the Jetson Tx2, the Jetson AGX Xavier and the Jetson Xavier Nx are used as the hardware backend for GeNN on GPU. As an CPU-only mode is supported in GeNN, this particular implementation can be tested on a variety of hardware solutions, such as desktop PCs and embedded systems. GeNN on CPU is additionally included to enable a fair comparison to the performance of the NEST simulator, which are both run on a single processor core of an AMD Ryzen3700x. This PC has 32 GB of RAM and contains an Nvidia RTX2070 GPU. How the different hardware solutions and simulators of this benchmark are related to PyNN by custom interfaces is visualized in Figure 5.6. The WFA, described in subsection 5.2.1 is used as a test scenario which is modeled using the respective PyNN interfaces for each tested system.

[3]`https://github.com/genn-team/pynn_genn`

[4]For a comparison of the Jetson boards see: `https://www.fastcompression.com/blog/jetson-benchmark-comparison.htm`

Differences Regarding the Hardware-related Implementations

The implementation via PyNN enables the use of the same model for all simulators. However, this means that it may not be possible to exploit all the strengths and unique features of the simulators, as also mentioned in [59]. It would probably be possible to develop an implementation in each of the systems that make optimal use of the specific strengths and thus produces better results than general modeling using PyNN. In the end, modeling in PyNN as well as in the native language of the backend has advantages and disadvantages. However, a central goal of this benchmark is a comparison of different technologies for simulating SNN, that is as fair as possible. Based on this, it was decided to use a uniform model through PyNN. Although, due to PyNN, basically the same model is used, the simulations are slightly adapted to the respective systems. This is necessary as the back ends implement slightly different subsets of functions and models supported by PyNN and the code could not be run on all systems without minor modifications. Furthermore, the use of an identical model on all systems leads to the problem that some technical specifications do not fit at all. To ensure a high degree of comparability, these considerations lead to two opposing design goals of the benchmark; using a similar model if possible, and exploiting the individual features of each system. Hence, an important design decision is to what extent the models may differ from each other.
Most simulators and hardware solutions for simulating SNN update the neurons' states at a fixed time step [427]. A big difference in the implementation of the simulators is thus the step size. Both GeNN and NEST run the simulations with the step size 0.1 ms, as it is common in a neuro-scientific context [57], but for SpiNNaker 1 ms is used. Adjusting the step size, no matter in which direction, would strongly distort the results. The use case used, described in subsection 5.2.1, is to solve path planning in a reactive and fast way, so slowing down NEST and GeNN would have an extreme impact. Consequently, all simulations in this benchmark are carried out at the default time step of the respective system. The different step sizes also require an adjustment of total simulation times, so that the simulations, the wave traversing the network as outlined in subsection 5.2.1, can run on the different platforms all the way through. The implementation of SpiNNaker requires a longer simulation time as NEST and GeNN, thus, different simulation times were chosen for the systems. The model implementa-

	NEST	SpiNNaker	GeNN
step size	0.1 ms	1 ms	0.1 ms
neuron model	`IF_cond_exp`	`IF_cond_exp`	`IF_cond_exp`
weights	unscaled	scaled	unscaled
spike source	DC source	SpikeSourceArray	DC source

Table 5.3.: Differences of the model implemented for NEST, SpiNNaker and GeNN. Table adapted from: (Steffen et al. 2021b; Koch 2020)

tion is based on the work of (Steffen et al. 2020b), which was originally written for NEST. In Table 5.3, all system-specific adaptations are listed. The NEST model used for the benchmark differs slightly from the original implementation regarding the neuron model. In (Steffen et al. 2020b) the neural WFA is realized with `IF_cond_alpha` a LIF model which embodies an α-function to characterize the postsynaptic potential of the neuron. As this model is, at least at the time of performing the benchmark, not supported in sPyNNaker, it was replaced by another LIF neuron, `IF_cond_exp`, where the postsynaptic potential is described as exponentially decreasing.
Since SpiNNaker represents synaptic weights by 16-bit integers the weights have to be converted by a bit shift [57]. The method used for the bit shift does not allow the maximum weight used in the original implementation. Therefore, the original `w_max` $= 4000.0$ µS is replaced by the highest possible value in SpiNNaker, `w_max` $= 63.0$ µS. But this change creates consequential issues, as STDP with additive weight dependence is used for the learning phase of the WFA and `w_max` $= 63.0$ µS is too low to induce the necessary adaptation of synaptic weight required to generate a purposeful SVF. As a workaround, the previously downscaled weight is scaled up by a factor of $f_{scale} = 4000.0/63.0$ post-simulation. This type of re-scaling introduces a small error into the system, as a term introduced by the STDP rule cannot be correctly displayed. It is also possible to sample the weights before and after the simulation and thus calculate Δ_w and scale it independently. It would allow a correct weight scaling but add significant overhead in the last phase of the algorithm, the pathfinding. However, as the results are only minimally distorted by the re-scaling it has been used here.
Lastly, the NEST implementation in (Steffen et al. 2020b) applies a DC to the target neuron to initiate the neural wave. Thereby the membrane potential is increased and the respective neuron emits an action potential, a spike. As sPyNNaker does not support DC sources the generation of the initial spike is implemented by `SpikeSourceArray`, a neural population that is connected to the respective nodes through projections. With this realization, the membrane potential is not increased, but a spike is triggered directly. This has the benefit that the algorithm can be started directly at the beginning of the simulation time. To get a similar behavior when using the DC source, it is set to 1000 mV.

5.2.3. Metrics

As it is not purposeful to compare parallel architectures, like GPU and neuromorphic hardware, with metrics used for benchmarking von Neumann architectures [409], metrics better suited for this use case are introduced. The metrics were chosen to emphasize the benefits and drawbacks of the platforms and give an indication of their usability for a robotics use case:

- The ***simulation time*** gives information about the performance of the system under a realistic workload. In particular, in path planning, where calculation speed is crucial, this indicator is of great significance. However, not only the time required for execution is measured, but also for loading and compiling.

- The average energy needed for a run is used to determine the ***energy consumption***, which is becoming an important factor in robotics, especially for mobile robots. The data generation regarding the energy consumption is done externally and is performed by a consumer-grade power meter, more precisely an energy meter of type Voltcraft 4000. The logger has a resolution of > 0.1 W and an accuracy of +/- 1 % and enables the local storage of timestamped data.
- The ***path length*** is an interesting indicator as it gives information about the different STDP implementations. It is investigated whether the simulators when given the same initial weights have different weights after learning. Since all neurons that are part of the path are stored in a list, the list's length can be used to determine the path's length. Furthermore, it is investigated whether the path length varies for the same map on a simulator and the errors that are caused by the bit shifting and the scale-up on SpiNNaker are observed here.
- The allocation of ***hardware resources***, in particular memory usage and CPU as well as GPU allocation of the Desktop PC and the Jetson boards. Logging software[5] is used for this purpose and it applies, in addition to the memory usage in the case of the Jetson boards and the PC also to CPU and GPU.

5.3. Experiments and Results

The evaluation of the WFA on NEST, SpiNNaker and GeNN is strongly based on the metrics introduced in subsection 5.2.3. To examine the different hardware solutions in terms of their ability to scale up, the maps of the original implementation of (Steffen et al. 2020b), of size $20 \times 20 \times 20$, are enlarged. Because the network is a representation of the environment, as the maps are enlarged the number of neurons increases cubically and the number of synapses exponentially. The maximum map size of the systems is limited by the available memory of the used hardware. For the SpiNN-5 board, the SDRAM constrains the number of simulated neurons. A maximum map size of $40 \times 40 \times 40$ can be reached, which corresponds to a network with 64 000 neurons and 374 400 synapses. In the case of the SpiNNaker, the extremely increasing simulation time is another limiting factor. Regarding GeNN, its simulation on GPU, only supports a map size up to $30 \times 30 \times 30$, using 27 000 neurons and 156 600 synapses. This holds as well for

[5]For the Desktop PC CPU and memory data is logged with glances (`https://nicolargo.github.io/glances/`) and GPU data with nvidia-smi(`https://developer.nvidia.com/nvidia-system-management-interface`). For the memory usage of the jetson boards a logging script based on jetson-stats(`https://github.com/topics/jetson-stats`) is used.

Jetson TX2 and Jetson Xavier Nx, which both have roughly 8 GB shared memory between GPU and CPU. In contrast, Jetson AGX Xavier has a shared memory of 16 GB, thus, maps can be scaled up to $33 \times 33 \times 33$, with 35 937 neurons and 209 088 synapses. The CPU-only version of GeNN, as well as NEST, supports theoretically map sizes up to $55 \times 55 \times 55$, by using the PC's 32 GB RAM. For NEST, however, this implies unrealistically long simulation times for maps larger than $35 \times 35 \times 35$.
Due to the low frequency of the power meter, described in subsection 5.2.3, the power draw is under-sampled and consequently the measurements regarding energy consumption are imprecise. This was counteracted by redundant measurements with subsequent averaging of the readings, which, however, cannot fully compensate for the poor temporal resolution. Hence, these results must be interpreted with caution, and conclusions about the systems are thereby compromised.
All experiments performed in this section, are simulated on the map IV from (Steffen et al. 2020b), as this is the most complex one.

5.3.1. Simulation Time

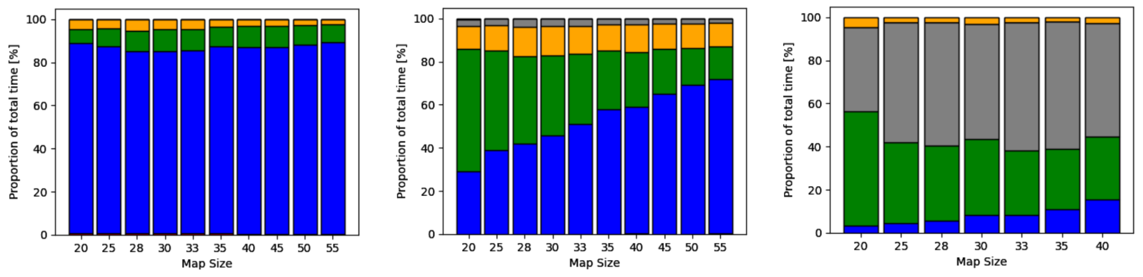
For the evaluation regarding simulation time, the median, as it is quite robust in case of outliers, and the standard deviation is used. This is done by wrappers for the functions `create neurons`, `create synapses`, `simulation`, `build SVF`, `compilation`, `load simulation` and `finding path`. Thereby, the timestamp is saved at the start and end of each function and the delta is its execution time. To determine the total time, the individual times of the functions are summed up for every run. There are significant deviations regarding

GeNN					**SpiNNaker**	**NEST**
Tx2	Xavier Nx	AGX Xavier	RTX 2070	CPU	SpiNN-5 Board	CPU
353.64	227.51	143.63	36.76	29.77	162.46	35.20

Table 5.4.: Simulation times in seconds for all hardware and software solutions for map size $30 \times 30 \times 30$. Table adapted from: (Steffen et al. 2021b; Koch 2020)

the initialization of neurons and synapses, within the simulation software of the three systems. While they are initialized at the definition for all three systems. In sPyNNaker and PyNN interface for GeNN (PyNN GeNN) synapses are instantiated again in the `run()` function, due to execution in C++ or CUDA and placement onto the vertices of the MachineGraph respectively. Furthermore, for sPyNNaker, the function `run()` includes loading and running of the simulation while in PyNN GeNN, compilation and running of the simulation. Hence, the time interval for the individual functions cannot be determined solely by the wrapper. However, if a simulation is run for 0 seconds, loading is triggered in sPyNNaker and compilation in PyNN GeNN. In the case of sPyNNaker, loading is restarted when the simulation of the WFA is carried out. Thus, to avoid counting the loading time twice, timestamps of the sPyNNaker logs are used.

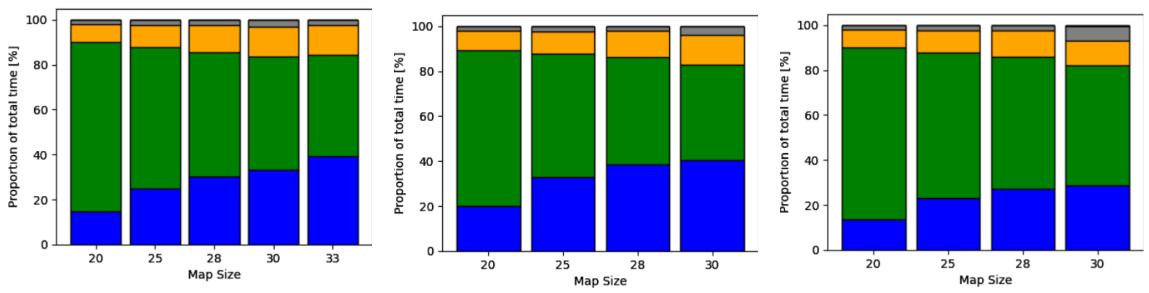
By comparing the simulation time on the largest map supported by all sys-



(a) SpiNNaker (b) GeNN on CPU (c) NEST

Figure 5.7.: Simulation time broken down for individual functions on (a) SpiNNaker, (b) GeNN on CPU and (c) NEST. The individual functions are colored. In blue `create synapses`, in green `load simulation`, in orange `simulation` and in gray `build SVF`. The functions, `create neurons`, `compilation` and `finding path`, have such a small share that they are not visible in the diagram. Image source: (Steffen et al. 2021b)

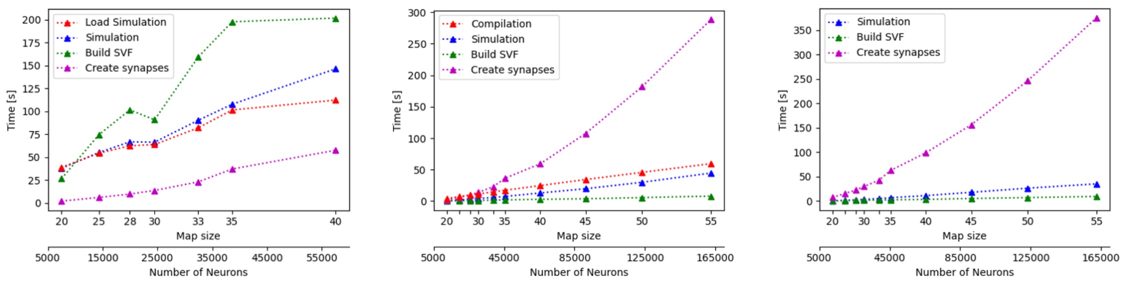
tems ($30 \times 30 \times 30$), as shown in Table 5.4, major differences can be determined directly. Contrary to all expectations, the implementations for the desktop PC were the most successful, concerning simulation time. The fastest execution time is achieved by GeNN on CPU, followed by NEST and GeNN for GPU on the Desktop PC. However, it is a relatively large difference of more than 5 s between GeNN on CPU and NEST, which in turn is very close to GeNN for GPU. Overall, GeNN on a Tx2 came off as the slowest with SpiNNaker in the second last place. However, the entire Jetson boards series performed surprisingly poorly, which can be seen in more detail for different map sizes in Table C.1 in Appendix C. In Figure 5.7 and 5.8 the share of individual functions in the total simulation time is



(a) Jetson Xavier AGX (b) Jetson Tx2 (c) Jetson Xavier Nx

Figure 5.8.: Simulation time broken down for individual functions on (a) Jetson AGX Xavier, (b) Jetson Tx2 and (c) Jetson Xavier Nx. Colors refer to the same functions as in Figure 5.7. Image source: (Steffen et al. 2021b)

visualized. While Figure 5.7 deals with a cross-simulator comparison, Figure 5.8 compares GeNN on different Jetson solutions. In Figure 5.7 SpiNNaker, NEST and GeNN on CPU are included. GeNN is shown on CPU, as this allows a larger map size which increases the visibility of trends. The most striking observation is that in NEST the synapse creation (`Create synapses`) accounts for the largest share by far. This is also true for GeNN on CPU in the case of large maps, but for smaller maps the time portion of `Compilation` is predominant. Regarding SpiNNaker, the creation of synapses is only a small portion of the total time, as `Simulation`, `Load Simulation` and `Build SVF` make up most of the time. For SpiNNaker this delay can be explained by communication overhead with the host, as several graphs need to be built and validated, and also it needs to be checked that all cores are ready. This communication between the Host and the SpiNNaker board, as visualized in Figure 5.1, is quite elaborate. The board needs to map the chips, the host creates an application- and machine graph, and subsequently the machine graph is loaded onto the board [410]. The biggest surprise of the benchmark was probably that GeNN performed relatively poorly on the Jetson boards. This discrepancy becomes very clear when observing that the simulation time of the map size $30 \times 30 \times 30$ on the RTX2070 is still significantly shorter than on each of the Jetson boards. It is, however, quite obvious, when



(a) SpiNNaker (b) GeNN on CPU (c) NEST

Figure 5.9.: Scaling Properties for SpiNNaker, GeNN & NEST. The median of the simulation time broken down for different functions is plotted in s for map sizes from $20 \times 20 \times 20$ to $55 \times 55 \times 55$ except for SpiNNaker, as the map size is limited to $40 \times 40 \times 40$. Image source: (Steffen et al. 2021b)

looking at the distribution of the simulation time on the individual functions and considering which of them are processed on CPU and GPU. From the functions shown in the diagrams in Figure 5.8, only `Simulation`, shown in orange, is processed on the GPU. Both `Create synapses` (blue), `Load Simulation` (green) and `Build SVF` (gray), are processed on CPU. Unsurprisingly, however, from the Jetson boards the AGX Xavier has the best total simulation times, followed by the Xavier NX. The only one of the Jetson boards that beats the simulation time of the SpiNNaker is AGX Xavier. The diagrams in Figure 5.9 visualize the scalability of SpiNNaker in (a), GeNN in (b) & NEST in (c). The simulation time of NEST increases exponentially with the map size, thus the number of neurons and synapses within the network. This is also true for the CPU-only implementation of GeNN and AGX Xavier and Tx2. However, for the GeNN implementation on the Xavier NX and the RTX2070, simulation time increases linearly. Regarding SpiNNaker, scalability is less clear. There are indeed linearly increasing segments here, but there are also large jumps such as between from map size $28 \times 28 \times 28$ to $30 \times 30 \times 30$. It is thereby important to note that, as well for NEST and GeNN, network construction (`create neurons` & `create synapses`) takes up a large part of the total time, as seen in Figure 5.7. For SpiNNaker, however, this is not a big factor and is overshadowed by loading (`load simulation`) and learning (`Build SVF`).

5.3.2. Energy Consumption

Due to the low frequent logging of the power meter, very few data points are available, thus an estimate of the total energy consumption is obtained by linear interpolation. To compensate for inaccuracies, several runs are measured and an average is taken. Results of all map sizes can be found again in Table C.1 in Appendix C. Additionally Table 5.5 shows an extract of the results for the map size $30 \times 30 \times 30$. The energy consumption of a system depends on two factors,

GeNN					**SpiNNaker**	**NEST**
Tx2	Xavier Nx	AGX Xavier	RTX 2070	CPU	SpiNN-5 Board	CPU
16762.27	24054.32	7169.49	3817.19	3125.71	13593.85	3089.95

Table 5.5.: Average energy consumption in J per run for all hardware and software solutions for map size $30 \times 30 \times 30$. Table adapted from: (Steffen et al. 2021b; Koch 2020)

total simulation time and energy efficiency. As can be seen in Figure 5.9, the total time per run increase with an upscaled map size for GeNN, SpiNNaker and NEST. Consequently, energy consumption is indirectly influenced in comparable dimensions. At first glance, the execution on the desktop seems to achieve good results, but this is deceptive when the simulation times are taken into account. About energy efficiency, the Jetson boards are the most efficient realization and NEST by far the worst. However, this is not reflected in the total energy consumption, thus, when evaluating Table 5.5 the high simulation times of the Jetson boards must be taken into account. A similar effect occurs when comparing the results for RTX 2070 and GeNN's CPU-only version, although the former is more energy efficient its energy consumption is higher. The results of the SpiNNaker are also strongly affected by large simulation times.

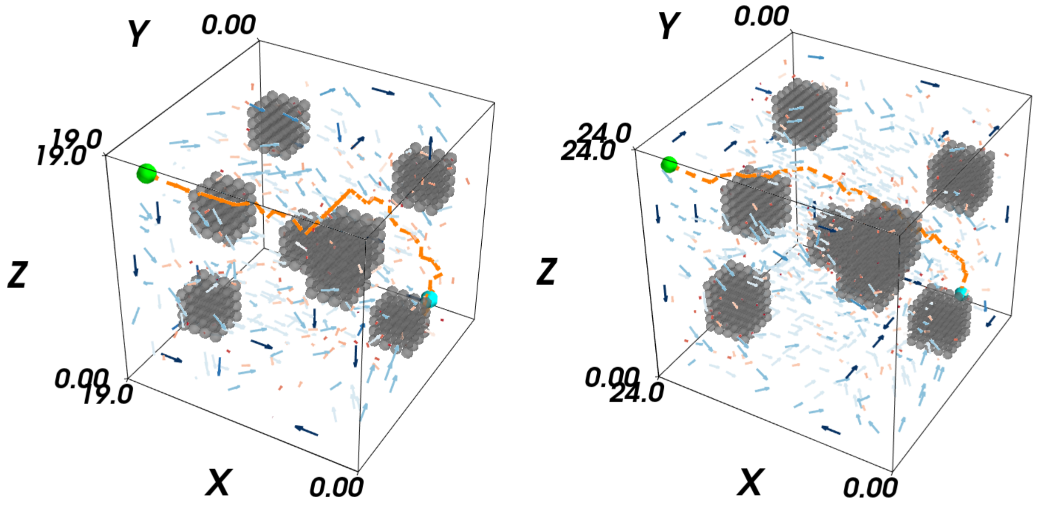
5.3.3. Path Length

The length of a resulting path depends on the network's synaptic weights after STDP takes place, as the WFA finds a path by traversing the SVF, which is an interpretation of the network's weights. To assure comparability, the same initial values should be used for the weights within each implementation. However, this was only feasible for NEST and GeNN, as the bit shifting leads to rounding errors in the SpiNNaker platform, which is further amplified by the subsequent upscale. As a consequence, a fair comparison is only possible for NEST

GeNN					**SpiNNaker**	**NEST**
Tx2	Xavier Nx	AGX Xavier	RTX 2070	CPU	SpiNN-5 Board	CPU
49	48	49	46	49	50	48

Table 5.6.: Path length for all hardware and software solutions for map size $30 \times 30 \times 30$. Table adapted from: (Steffen et al. 2021b; Koch 2020)

and GeNN.
During the simulation a list is generated containing all nodes of the path, consequently, its length is the length of the path. The values stated in Table 5.6, are medians of the path lengths taken over all runs on a particular map and system. By also considering the standard deviation, it is investigated if the path length differs between individual runs for identical conditions. As expected the path length generated by the SpiNNaker board varies from the other implementations. Also, it can be seen that for NEST and GeNN the final weights after learning are quite similar, as for the Xavier Nx the resulted path length is identical to the NEST implementation. Nevertheless, a major inconsistency is noticeable between the Jetson boards, this is even more visible in Table C.1 in Appendix C. Another irregularity is shown in Figure 5.10. As discussed in (Steffen et al. 2020b) the WFA does not guarantee an optimal path and therefore surprising results can arise. The path generated on a map of size $20 \times 20 \times 20$ as shown in Figure 5.10(a) has more detours than for the map with size $25 \times 25 \times 25$, thus the path is shorter on the larger but otherwise identical map.
Furthermore, on the same maps diverging path lengths are sometimes generated not only across boards but also for different runs on the same board. Since this behavior was very unexpected, it was ensured that weights were not compromised during compilation or initialization on the GPU, by checking them during run time. Consequently, the diverging path lengths are caused during the simulation of the WFA. A possible explanation for this odd behavior is that floating point numbers are non-associative. Hence, during parallelized additions on GPU, rounding occurs slightly differently depending on the termination order of the threads included.

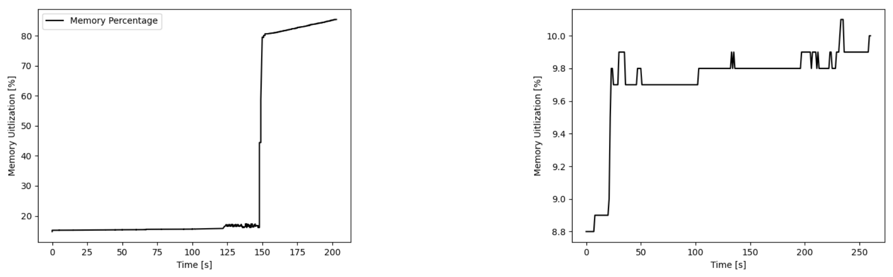


(a) (b)

Figure 5.10.: Visualization of the path generated with GeNN on CPU, in (a) for map size $20 \times 20 \times 20$ and in (b) for $25 \times 25 \times 25$. The path is colored in orange, the start neuron in green and the target in cyan. It is visible that the path on the larger map is shorter than on the smaller one. Image source: (Steffen et al. 2021b)

5.3.4. Hardware Resources

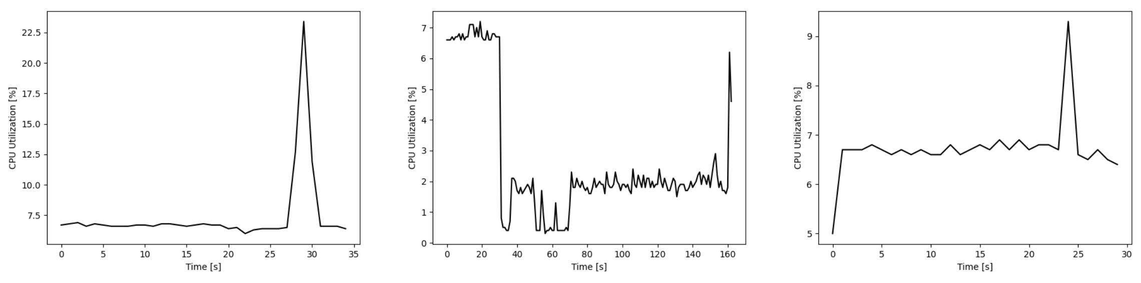
As the resource consumption, both in terms of memory and CPU usage, for the respective solutions is very constant on different maps and map sizes, one was used as an example for the investigation, provided regarding the memory usage in Figure 5.11 and CPU usage in Figure 5.12. The graph in Figure 5.11(a) shows



(a) AGX Xavier (b) SpiNNaker

Figure 5.11.: Memory requirements for (a) AGX Xavier as an example of GeNN and (b) SpiNNaker as neuromorphic hardware. Tests are performed on the map IV from (Steffen et al. 2020b) with the map size $33 \times 33 \times 33$. Image source: (Steffen et al. 2021b)

the memory usage in percent for the AGX Xavier as an example for all GeNN implementations, as they have shown very similar trends. During loading and compiling, the plot shows a very slow and steady upward trend regarding memory usage. This is followed by a dramatic jump at the start of the simulation, which is succeeded by another slow and steady increase during the simulation, but with a larger slope than before. The development of memory usage on the SpiNNaker is in stark contrast to that of the GeNN implementations. A major leap in memory usage occurs very early on, during the generation of the synapses. A smaller jump is also present when loading the simulation. In general, the graph shows more ups and downs. In Figure 5.12 the development of the CPU usage



(a) GeNN on GPU (b) SpiNNaker (c) GeNN on CPU

Figure 5.12.: CPU utilization for GeNN on (a) GPU, (c) CPU and (b) SpiNNaker. Tests are performed on the map IV from (Steffen et al. 2020b) with the map size $30 \times 30 \times 30$. Image source: (Steffen et al. 2021b)

is plotted in percent for (c) GeNN on CPU, (a) GeNN on the RTX2070 and for (b) SpiNNaker. The plot in Figure 5.12(a) shows that the memory requirement on the RTX2070 is constantly low during most of the run time, this is only interrupted by a single pronounced spike between the loading and compiling of the simulation. For the CPU-only implementation of GeNN, in (c), the CPU usage is consistently higher than for the GPU implementation in Figure 5.12(a). This was expected as the simulation runs on a single-threaded CPU backend. Moreover, the plot in Figure 5.12(a) shows a strong increase at the beginning of the run time and a relatively large spike shortly before the end. The large spike occurs during the simulation of the WFA itself. Furthermore, the CPU consumption in Figure 5.12(a) and (c), is similar for most of the time in both systems. However, it is very noticeable and surprising, that the spike has a much higher increase on the CUDA backend in (a), to almost 23 % compared to less than 10 % on the single-threaded CPU in (c). The plot regarding the SpiNNaker version in Figure 5.12(b), shows a significant increase in memory consumption while the creation of the synapses as well as the generation of the SVF by the function `Build SVF`.

5.4. Discussion

In subsection 5.2.3, metrics are defined as suitable for a purposeful benchmark of parallel hardware. It introduces four indicators as the basis of evaluation in

section 5.3. Thus, simulation time is investigated in subsection 5.3.1, energy consumption in subsection 5.3.2, path length in subsection 5.3.3 and lastly hardware resources in subsection 5.3.4. The path length is of relevance as it provides information about how learning through STDP is realized in the different realizations. As path length only varies slightly between the NEST, GeNN and SpiNNaker implementation, it is assumed that there are no major differences regarding the underlying processes of STDP. In general, this mechanism also seems to produce consistent results, as the implementations of NEST, SpiNNaker and also GeNN, in regards to the CPU-only implementation or on RTX2070, produce identical path length when run multiple times. This observation is consistent with the results of [57; 58] that NEST, SpiNNaker and GeNN produce very similar results. However, in [57; 58] only static synaptic weights are used, hence, this thesis is generalized to learning with STDP in the presented benchmark. Similar observations are also made in [59] for GeNN and SpiNNaker, however only for CNNs which were transformed to SNN after learning took place.
GeNN executed on all Jetson boards generates different path lengths for multiple runs under identical circumstances. This result is justified by the non-associative floating point numbers used for GPU, which may cause varying results when additions are parallelized, as introduced in subsection 5.1.1. As stated in [414], this makes it very difficult to compare calculations from CPU and GPU. Similar observations, that results may differ between simulation runs in GeNN, have been reported in [12]. It is also noteworthy that this benchmark only simulates a single spike-wave. Hence, the model is more susceptible and even small deviations can have a big impact.
Furthermore, it is very interesting that the good performance regarding simulation time for GeNN on CPU, as stated in [59], are reproduced in this benchmark. Also, in [57; 58; 59] it was stated that neuromorphic hardware has a hard time efficiently initializing and loading networks. This observation was confirmed here for SpiNNaker and is explained by the communication overhead.

5.4.1. Context Analysis

As well the SpiNNaker as the GeNN implementation, when executed on the Jetson boards, induce a communication overhead, as stated for SpiNNaker in subsection 5.1.1. In contrast NEST and GeNN, run on the PC, are compiled locally, thereby eliminating the need for communication between host and device. While this fact is obvious, its implications are significant. This section discusses the aftereffects of the communication overhead as well as other factors that affect the validity of the results presented in section 5.3.

Assignment of Tasks to CPU and GPU

The relatively poor results achieved by GeNN on GPU, especially when executed on the Jetson boards, are mainly because many of the functions used to simulate the WFA are performed by the CPU. This concerns, compiling, loading and synapse creation. Only the actual simulation is run on CPU. As the Jetson boards are designed for GPU heavy applications, they only include a comparatively lightweight ARM-based CPU cluster. The maximum frequency that the CPUs of the Jetson boards can achieve, varies between 1.9 GHz for the Xavier Nx and 2.26 GHz for the AGX Xavier. In contrast, consumer CPU have a higher maximum frequency than embedded systems. Consequently, the results reflect a large discrepancy between the simulation times for GeNN on desktop PC and the Jetson boards. Furthermore, the comparatively good results of the CPU-only implementation are reasoned by the fact that neither additional transfer is necessary between host RAM and GPU, nor the generation of CUDA code.

Complex Synapse Creation Prevents Scaling Up

The NEST implementation is completely unaffected by communication overheads, as no data is compiled or loaded on an external device. However, the time required to generate the synapses and neurons is significant and exceeds the CPU-only version of GeNN. Consequently, NEST scales particularly poorly, as larger maps require more neurons and synapses to be created. Synapse creation is performed on CPU for all software solutions. In GeNN, PyNN projections need to be instantiated for this, once for the initial creation and also during simulation. Hence, when comparing the generation of synapses between NEST and GeNN, the compilation time of GeNN must also be considered. However, despite taking the compilation time into account, synapses and neurons are generated faster than in NEST. This is crucial for large networks, where NEST needs up to 20 seconds longer to generate synapses, compared to GeNN's CPU-only version. This advantage of GeNN has a big impact, as although NEST has shorter simulation times than GeNN for all map sizes, the total time of GeNN is better, especially for large networks. Furthermore, for GeNN scaling up the network size increases significantly the run time for synapse creation and not for compilation and simulation. This implies that large networks are handled better on the native frontend than with PyNN GeNN and is not surprising as the compiler language C++ is generally faster than the interpreter language Python [435].

Weight Transfer Between Host and Device

On closer inspection, it becomes apparent that synapse generation on the SpiNN-5 board takes a similar amount of time as for GeNN on CPU. This is only overshadowed by the extremely long times for loading, simulating and generating the SVF. However, delay times regarding loading and simulation are due to the communication overhead [410], outlined in subsection 5.1.1. Moreover, building up the SVF takes place on the CPU and requires extracting the synaptic weights from the simulation. For SpiNNaker, this means that the weights are transferred from the local memory of the SpiNNaker board to the host's RAM via a 100 Mbit

Ethernet cable. In contrast, for NEST and GeNN on CPU, weights can be loaded directly from the system RAM. Thus, generating the SVF requires significantly more time for the SpiNNaker implementation than NEST and GeNN on CPU, as the weight transfer on SpiNNaker is about 10 times slower than on the internal data buses. In case of local execution of GeNN on GPU, the times for building the SVF increase slightly, since weights are loaded from the memory of the GPU.

5.4.2. Limitations and Implications

The biggest limitation of the presented results is regarding the measurement of energy consumption in subsection 5.3.2. As shown in subsection 5.1.3, some works in literature, such as [57; 58] use more sophisticated types of energy measurement. For example, system-integrated power draws or data evaluation using image recognition software. Since these possibilities are not so easily accessible here, the results regarding energy consumption are less meaningful. However, energy consumption was measured in a similar way in [59], as done within this chapter.
The benchmark emphasized that memory is a big factor, by limiting the number of neurons an SNN can include. Adding to this problem, memory on GPU is usually less than the RAM on a PC. For the Jetson boards memory is shared between CPU and GPU. Consequently, the boards enable only a relatively small maximum map size. As indicated in the sections before, the comparison of embedded systems and high-performance GPU on a PC is somewhat problematic. It was clear from the beginning that the GeNN implementation on the RTX2070 is faster and consumes more energy than on the Jetson boards. However, the investigation is purposeful as the exact difference between the two is not clear, and provides information about the state of development of embedded systems. In general, the analysis of GeNN is also somewhat impaired. As outlined in subsection 5.4.1, choosing PyNN GeNNfor implementation instead of GeNN's native frontend introduces an overhead, as synapses and neurons have to be instantiated costly in PyNN, before simulation.

6. Conclusion

In this thesis, a neural system was aimed to combine perception, motion control, obstacle detection and path adaptation. For this purpose, new brain-inspired technologies are used, which exploit the advantages that nature has over technology. The research questions 1, 2 and 3 were derived from this abstract goal, the exploration of which represents the core of this work. Therefore, the focus was not on creating a holistic system that solves the objective perfectly, but on developing meaningful answers to the proposed research questions. The thesis is highly motivated by neural processes and mechanisms such as the human visual system and general research in neuroscience. Thus, it seems appropriate to note that strict compliance with biological principles was deliberately neglected. Instead, the motivation was to make findings from these theoretical research fields accessible for robotic applications, especially in an industrial context.

6.1. Summary

As this thesis is interdisciplinary, but intended for readers with a strong background in computer science the foundation in chapter 2 focuses on concepts within the field of biology and computational neuroscience. Thereby, Spiking Neural Network (SNN), Self-organizing Neural Network (SONN) and event cameras are motivated by their biological model in section 2.1, followed by their introduction in section 2.2. In section 2.1 biological navigation is also depicted, however, as there is not a well-established realization therefore yet it was neglected in section 2.2.
The challenges posed by the research questions in section 1.2 have all been addressed in the following chapters. In chapter 3, research question 1 is investigated, *"How can asynchronous event streams be optimally exploited for event-based stereo vision?"*. After consideration of monocular and stereo techniques for depth reconstruction, it is concluded that stereo vision is superior, also for technical applications and especially in the case of a static visual system. Furthermore, the research on the state-of-the-art confirms that for the processing of event-based data, no algorithms from the field of frame-based computer vision should be transferred. However, it is also clear that there is still a lot of potential concerning event-based processing. Subsequently, a method using self-organization, a brain-inspired technique introduced in subsection 2.1.2 and 2.2.2, is explored for event cameras. Thereby, the correspondence problem related to stereo vision is considered as a dimension reduction. The 4D data, which consists of the 2D data

of two event streams, embodies depth information about the recorded objects. This underlying 3D structure can be recovered by the use of a Self-Organizing Map (SOM), a topology-protecting technique for dimension reduction. The basic method is initially evaluated in simulation and was then transferred to an event camera, the Asynchronous Time-Based Image Sensor (ATIS). The initial technique suffered a few weaknesses. Firstly, the actual matching was very slow, as it considered all neurons when determining a Best Matching Unit (BMU) and secondly, the outer layers of the SOM performed poorly. Lastly, a time-intensive learning phase was necessary, whereby movements should take place evenly distributed over the work cell, so neurons may learn their respective association. Consequently, the integration of several improvements was necessary before a successful application on an event-based stereo setup was obtained. This includes receptive fields and pre-learning of an enlarged work cell which was generated by a simulated version of the workspace and sensor setup. The chapter closes with a discussion, whereby attention is drawn to the fact that, despite the promising results in simulation, the results on the event cameras could still be improved.
In chapter 4, research question 2 is investigated, *"Is it feasible to use the high-dimensional configuration space, which requires huge amounts of neurons, for neural path planning?"*. Hereby, the state-of-the-art is divided into three parts. Conventional path-planning methods are discussed as they are on the one hand part of the proposed method and on the other hand applied for evaluation. The former regards the Wavefront Algorithm (WFA) and Dijkstra and the latter sample-based methods as Probabilistic Road Map (PRM) and Rapidly Exploring Random Tree (RRT). Subsequently, brain-inspired methods for robotic control are investigated. However, most SNN-based approaches operate in 2D. As a consequence, one example [39] has been transferred to 3D environments. As the results were not sufficient for an online demonstrator an alternative using self-organization was developed. However, the 3D path planner using spiking neurons was applied as the test candidate of the benchmark in chapter 5. The state-of-the-art is concluded with a section on path planning in a reduced configuration space (C-space). This forms the basis for the core part of this chapter. Path planning is thereby performed in a subspace of the robots C-space. It is generated by a Growing Neural Gas (GNG) trained on simulated robot trajectories. A preliminary in-depth analysis of several network types was carried out, to ensure a good network structure for this approach. To allow dynamic obstacle avoidance bidirectional Lookup Table (LUT) are used to map objects from the task space T into the output space of the trained GNG. The thereby created cognitive map is a representation of a subspace of the robot's C-space. As it is significantly smaller than the complete C-space, it allows the application of an optimal planner. For this purpose, the WFA and Dijkstra have been tested. Because Dijkstra considered edge weights, it was superior in the evaluation. As the approach was very performant it was successfully applied to an online demonstrator. The most important part of its evaluation is the comparison to sample-based planners. While sample-based planners generate non-deterministic trajectories, the proposed method creates an optimal path. It could be shown that it is faster than RRT and PRM and comparable to the Rapidly Exploring Random Tree Connect (RRT-C).

In chapter 5, research question 3 is investigated,*"How can parallel hardware help to exploit the advantages of SNN?"*. Therefore, state-of-the-art parallel hardware solutions and simulation tools for SNN are presented. Afterward, related benchmarks are discussed and a lack of benchmarks considering robotic use cases especially regarding Graphics Processing Unit (GPU)-based SNN simulation is observed. For the benchmark, a neural implementation of a 3D WFA is used. On the one hand, this reflects a very typical robotic use case and on the other hand the widely used Spike-Timing-Dependent-Plasticity (STDP) is used for learning in the SNN. Both of these factors increase the significance and generalizability of the benchmark. Thereby, the simulator NEural Simulation Tool (NEST), the neuromorphic hardware spiking neural network architecture (SpiNNaker) and the GPU-based representative GPU-enhanced Neuronal Networks (GeNN) were used to cover a wide field of possible candidates. Four metrics were used for evaluation purposes, the relevance of which is derived from related work. The evaluation of this benchmark is based precisely on these metrics. The GPU library GeNN allows a variety of GPU-based technologies to be tested, however, there have been no major surprises in the comparison of GPU-based technologies. In contrast to the previous chapters, there is a greater need to deal with the limitations of the benchmark and its implications. Thus, the final section is supplemented by subsection 5.4.2, which handles them, in particular regarding the measurement of energy consumption. Moreover, it is also necessary to evaluate the results and their significance in the context of the intended applications, which is done in subsection 5.4.1.

6.2. Discussion and Outlook

As shown in Figure 1.3, it would be very interesting to integrate the isolated parts of this thesis. The stereo vision method described in chapter 3 could be used for proprioception and replace the vision component of the path planner in chapter 4. However, the main problem with this and the reason that has prevented this is that the stereo vision setup has an occlusion problem when used in a shared workspace as done for the demonstrator of the path planning method. Depending on the orientation, either the robot could hide potential obstacles or the other way around. This issue, that concealment hinders the use of the stereo vision approach for the path planning demonstrator, could be overcome by extending the vision setup. Therefore, two options could be explored. Firstly, a setup of three event cameras, as the method in chapter 3 is not limited to a stereo input. Secondly, the application of sensor fusion for event-based sensors with, for example, point clouds opens great opportunities. This type of setup allows to combine the accuracy of point clouds with the reactivity of event cameras.
In regards to neural path planning, the method introduced in chapter 4 could be extended to include velocities, accelerations or torques similarly to [380]. Furthermore, continuous online learning regarding the SONN could further enhance

path quality and lead to shorter and smoother trajectories. This could be realized by continuing learning during execution. Hence, while path planning is already performed, synaptic weights are still updated and new neurons are inserted. However, this might cause previously learned information to get lost, as neurons and synapses are also deleted. Consequently, after some time a generally trained SOM would adapt to a specific task and likely not be able to perform trajectories initially. It would be interesting to investigate whether parts of the network can be frozen, to prevent overwriting.
It is noteworthy that the work of [377; 378] has a few similarities with the concept presented in this thesis. The authors apply a SOM to discretize an underlying 3D submanifold of a 4D input space adaptively, as described in chapter 3. As well in [377] as in [378] shown in Figure 4.6 a stereo setup of two cameras is used. In contrast, the presented approach, Figure 4.20, uses four stereo cameras. Another big difference between these methods and the approach presented here concerns the evaluation, as [377] and [378] are only tested in simulation.
Work in the field of spiking neurons, as (Steffen et al. 2019a; Steffen et al. 2020b; Steffen et al. 2020a), has been developed in the course of this dissertation. Unfortunately, there are not yet hardware solutions that allow their simulation in a reliable and performant way. Nevertheless, there has been a trend recently for these processors to be more accessible for research purposes and academia. As stated, for example, in [240], their optimization and further development are expected to have a significant impact on the research related to event cameras. A similar effect can be expected for spiking algorithms for robotic motion control.
Complete and optimal planners like WFA, Dijkstra or A* are not yet valid options for path planning in the high-dimensional C-space. However, this might change over time. Their execution, in the case of many Degree of freedom (DOF), is not feasible on conventional hardware, but, development in the field of neuromorphic computing and edge devices is rapid. If future research in SNN and neuromorphic hardware live up to the current expectations, optimal planners might apply to a full C-space, even for many DOF [407]. Apart from the level of development regarding neuromorphic hardware, their availability is currently an issue. Nevertheless, this is already starting to change, with neuromorphic solutions becoming more accessible for academic and research purposes. The approach presented in chapter 4 could also be transferred to SNN and neuromorphic hardware. As this was the original plan, the WFA was initially applied, as published in (Steffen et al. 2021c). The motivation was the parallelizability of a Breadth-first Search (BFS), which promised a fast and efficient future implementation with SNN. However, subsection 4.3.2 proved that the presented method benefits greatly from using Dijkstra's algorithm. Consequently, an SNN-based implementation realizing weighted connections, would be an interesting alternative. A respective example is provided in [47] by introducing axonal delays.

Appendix

A. Appendix for chapter 3

A.1. Sensor Specifications

(a)

(b)

Figure A.1.: Photos of the used sensors. (a) Prophesee Evaluation Kit Gen3 HVGA-EM [436] with ATIS architecture [251]. (b) RealSense^{TM} Depth Camera D435 [437].

Resolution	480×360
Pixel Pitch	$20\mu m$
Optical Format	$3/4inch$
Latency	$200\mu s$
Temporal Resolution	$1\mu s$
Field of view (H×V)	$56.3° \times 43.7°$
Dynamic Range	$120dB$
Interface	USB 3.0

Table A.1.: Specifications of the event camera Gen3 ATIS from Prophesee. Table source: [436]

Depth technology	Stereoscopic
Resolution of depth sensor	1280×720
Resolution of color RGB camera	1920×1080
Operating distance	$0.2m - 4.5m$
Depth Field of View (H×V)	$87° \times 58°$
Depth frame rate	Up to $90fps$
RGB frame rate	$30fps$
Interface	USB 3.0

Table A.2.: Specifications of the Intel® RealSense™ Depth Camera D435. Table source: [437]

B. Appendix for chapter 4

B.1. Constant Parameters for the SONN Analysis

The parameters are used for evaluating different SONN models, especially the quantitative analysis, in subsection 4.3.1 for 6D input. In Table B.1 the constant

$\#N_s$	η_{BMU}	η_n	λ	ζ	δ	ϵ	α	β	d	τ_{max}	AE_{max}
4	0.06	0.005	20	0.995	0.3	100	0.3	0.7	3	50	0.1

Table B.1.: n_s and n_e are the net' start & end size. The winner neuron's learning rate is η_n and its neighbors η_n. λ sets the amount of steps until a new node is inserted, ζ decreases the error counter and δ sets its ratio. The temporal context is balanced by α & β and the depth that indicates how many steps into the past are considered is defined by d. The number of samples considered for a segment is limited by τ_{max} and AE_{max} defines the maximum of the enclosed area. Table source: (Weyer 2021)

parameters for GNG-based models are given. The parameters,$\#N_s$-ϵ, are relevant for the GNG, MGNG, γ-GNG & SGNG. However, α & β are set only for MGNG & γ-GNG, and d only for γ-GNG. Lastly, τ_{max} & AE_{max} are only set for the SGNG. Respectively, in Table B.2 the parameters for SOM models are provided. Except

$\#N$	dim	λ	η	α	β	d
85x85 (7225)	2D	5.0	0.2	0.3	0.7	3

Table B.2.: $\#N$ is the pre-selected network size and dim the output dimension. The neighborhood radius is given by λ and the start value for the decaying learning rate is defined through η. How strongly temporal context is merged is balanced through α & β and the context depth by d. Table source: (Weyer 2021)

the context depth d, which is only relevant for the γ-SOM, all parameters are given for the MSOM & γ-SOM.

B.2. ROS 2 Components for Online Motion Control

Mainly responsible for the online control of the robot are the ROS 2 components */trajectory_execution_action_server* as well as */trajectory_execution_action_client*, as shown in Figure B.1. The server requests the trajectories from the client and forward them to the */joint_trajectory_controller*, which executes movement control. The point at which the information converges is the action client. For one, it re-

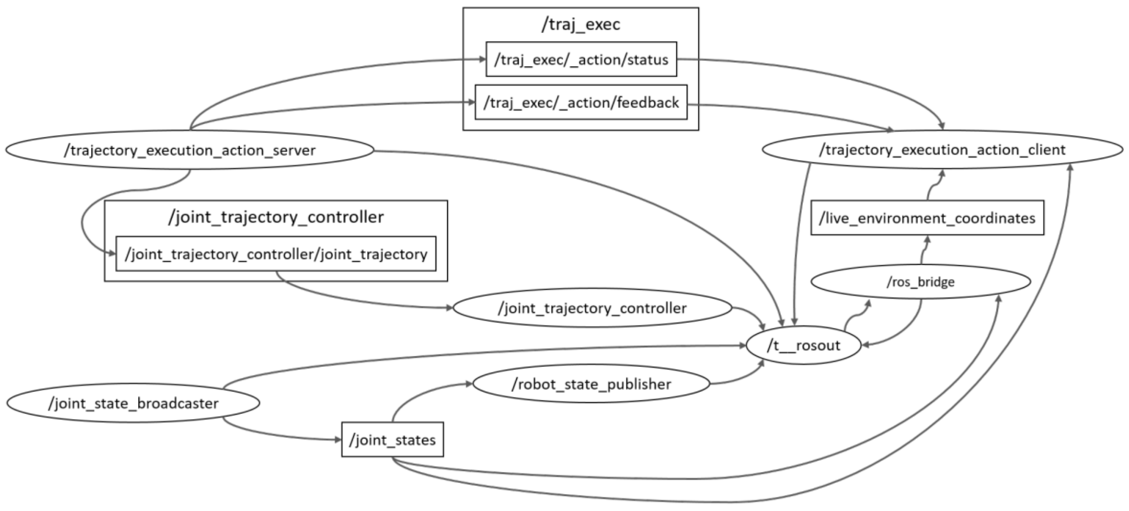
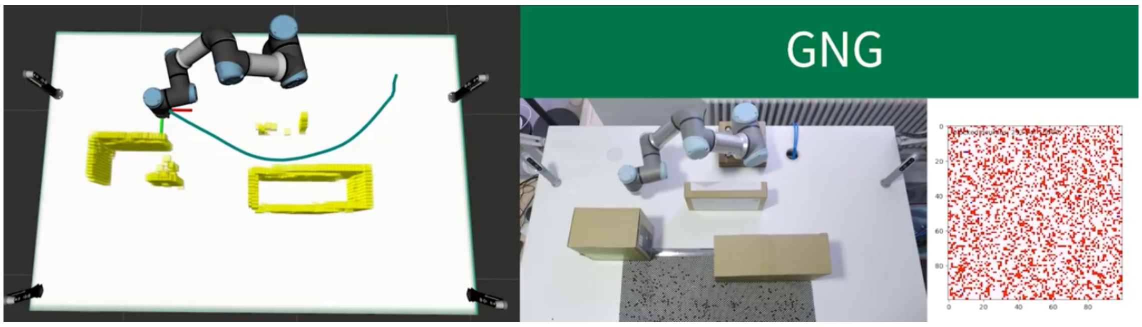

/traj_exec
/traj_exec/_action/status
/traj_exec/_action/feedback
/trajectory_execution_action_server
/trajectory_execution_action_client
/joint_trajectory_controller
/joint_trajectory_controller/joint_trajectory
/live_environment_coordinates
/ros_bridge
/joint_trajectory_controller
/t__rosout
/robot_state_publisher
/joint_state_broadcaster
/joint_states

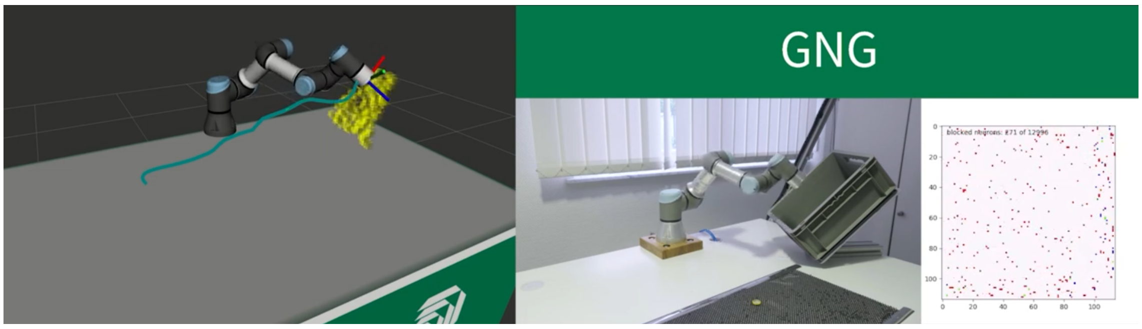
Figure B.1.: Overview of the ROS components responsible for online control of the robot. Image source: (Weyer 2021)

ceives information about current execution steps as well as the result from the server. Furthermore it obtains live data of the robot's joint states, and additionally, live environment coordinates of obstacles in form of voxel data from the ROS bridge. Lastly, the goal configuration, which can be set externally, is also known here. This information allows the client to plan a collision-free trajectory and transmit it to the controller via the server. During execution, collision checks are repeated regularly to ensure that the trajectory remains collision-free. If a collision would occur, the trajectory is aborted immediately, via a cancellation message, and a new collision-free trajectory is sent to the server.

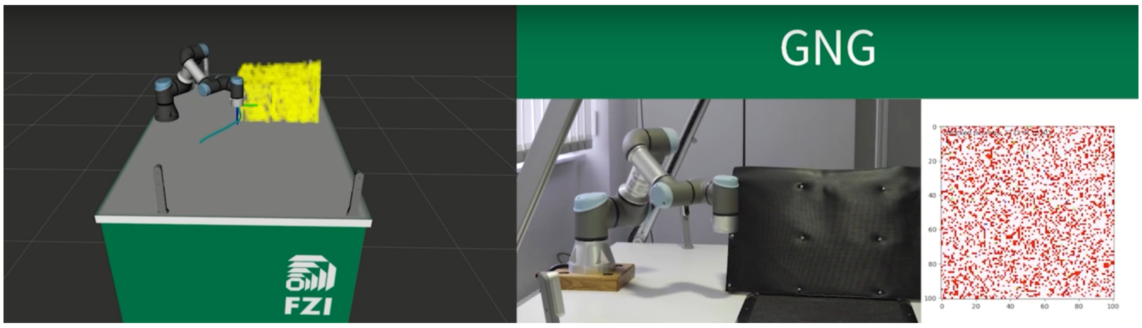
B.3. Special Features of the Path Planning Method



(a)



(b)



(c)

Figure B.2.: Photos and visualization of the demonstrator highlighting special features of the path planning method. (a) Path finding in a cluttered scene using the GNG. A with many big obstacles is a particular challenge with which sample-based planners as well as this approach using a SOM struggle with. (b) & (c) examples of applications that require the maintenance of certain configurations. In (b) a specific position of the wrist is necessary and in (c) an obstacle is circumvented at the same height without driving over it with the robot. Image source: `https://youtu.be/CEkVDDg9ORw`

C. Appendix for chapter 5

C.1. Additional Benchmark Results

In Table C.1 a complete listing of the results of Appendix C is given, regarding the benchmark for hardware solutions for simulating SNN. Simulation time is given in seconds, energy as an average per run in joules and the path length in amount of included neurons. All experiments are performed on map IV of (Steffen et al. 2020b). The source for Table C.1 is the supplementary material of (Steffen et al. 2021b).

system	map size	total time [s]	path length	energy [J]
GeNN on RTX2070	20	14.46	41.0	1201.82
	25	22.67	38.0	1376.56
	28	30.26	47.0	2728.85
	30	36.76	46.0	3817.19
GeNN on CPU	20	7.20	41.0	526.98
	25	15.15	35.0	1107.74
	28	23.11	47.0	2347.31
	30	29.77	49.0	3125.71
	33	43.29	46.0	4370.87
	35	63.59	55.0	5777.70
	40	98.67	58.0	10155.23
	45	163.36	60.0	16355.57
	50	262.22	69.0	24766.49
	55	409.27	79.0	39065.35
Jetson Tx2	20	114.22	41.0	143.54
	25	199.65	35.0	5213.75
	28	285.93	47.0	12894.51
	30	353.64	49.0	16762.27
Jetson Xavier Nx	20	86.86	34.0	3018.58
	25	137.98	37.0	5280.65
	28	186.83	46.0	1762.93
	30	227.51	48.0	24054.32
Jetson AGX Xavier	20	54.13	36.0	2254.65
	25	85.38	40.0	4358.44
	28	116.33	46.0	5521.26
	30	143.63	49.0	7169.49
	33	201.29	50.0	2347.63
SpiNNaker	20	64.62	32.0	13132.09
	25	128.50	49.0	23881.05
	28	171.26	44.0	14484.20
	30	162.46	50.0	13593.85
	33	259.03	53.0	21046.42
	35	324.96	55.0	24599.44
	40	368.47	72.0	30573.19
NEST	20	8.39	38.0	1861.54
	25	18.34	42.0	1908.04
	28	26.76	41.0	2453.65
	30	35.20	48.0	3089.95
	33	50.28	48.0	4294.51
	35	72.84	58.0	6126.01
	40	114.15	65.0	7641.54
	45	180.23	66.0	12266.63
	55	421.65	81.0	50946.67

Table C.1.

List of Figures

List of Tables

Bibliography

[1] Kortenkamp, David and Reid Simmons (2008). “Robotic Systems Architectures and Programming”. In: *Springer Handbook of Robotics*. Springer Berlin Heidelberg, pp. 187–206.

[2] Gat, Erann (1997). *On Three-Layer Architectures*. Tech. rep.

[3] Rieke, F. et al. (1999). *Spikes: exploring the neural code*.

[4] Maass, Wolfgang (1997). *Networks of Spiking Neurons: The Third Generation of Neural Network Models*. Tech. rep. 9, pp. 1659–1671.

[5] Grüning, A. and S. Bohte (2014). “Spiking neural networks: Principles and challenges”. In: *ESANN 2014*, pp. 1–10.

[6] Vreeken, Jilles (2003). *Spiking neural networks, an introduction*.

[7] Bi, Guo Qiang and Mu Ming Poo (1998). “Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type”. In: *The Journal of neuroscience* 18.24, pp. 10464–10472.

[8] Markram, Henry et al. (1997). “Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs”. In: *Science* 275.5297, pp. 213–215.

[9] Neftci, Emre O. (2018). *Data and Power Efficient Intelligence with Neuromorphic Learning Machines*.

[10] Furber, S. et al. (2014). “The SpiNNaker project”. In: *Proc. of the IEEE* 102.5, pp. 652–665.

[11] Davies, M. et al. (2018). “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”. In: *IEEE Micro* 38.1, pp. 82–99.

[12] Yavuz, Esin, James Turner, and Thomas Nowotny (2016). “GeNN: A code generation framework for accelerated brain simulations”. In: *Scientific Reports* 6.1, pp. 1–14.

[13] Furber, Steve and Steve Temple (2007). *Neural systems engineering*.

[14] Zenke, Friedemann and Wulfram Gerstner (2014). “Limits to high-speed simulations of spiking neural networks using general-purpose computers”. In: *Frontiers in Neuroinformatics* 8.SEP, pp. 1–15.

[15] Gallego, Guillermo et al. (2019). “Event-based Vision: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* abs/1904.0.

[16] Kaiser, Jacques (2020). “Synaptic Learning for Neuromorphic Vision – Processing Address Events with Spiking Neural Networks”. PhD thesis. Karlsruher Instituts für Technologie.

[17] Kelemen, Michal et al. (2018). “A novel approach for a inverse kinematics solution of a redundant manipulator”. In: *Applied Sciences (Switzerland)* 8.11.

[18] Koganezawa, K. (1994). “A fast method of solving inverse kinematics of redundant manipulators”. In: *IFAC Proceedings Volumes* 27.14, pp. 369–374.

[19] Kavraki, Lydia E. et al. (1996). “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE Transactions on Robotics and Automation* 12.4, pp. 566–580.

[20] LaValle, S. (1998). *Rapidly-exploring random trees : a new tool for path planning*. Tech. rep. Computer Science Department, Iowa State University.

[21] Schaal, Stefan, Christopher G. Atkeson, and Sethu Vijayakumar (2002). “Scalable Techniques from Nonparametric Statistics for Real Time Robot Learning”. In: *Applied Intelligence* 17.1, pp. 49–60.

[22] Schaal, Stefan (2006). “Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics”. In: *Adaptive Motion of Animals and Machines*. Tokyo: Springer, pp. 261–280.

[23] Naumov, V. et al. (2015). “Robot path planning algorithm”. In: *Int. J. Comput. Commun.* 9, pp. 96–99.

[24] Al-Jumaily, Adel and Cindy Leung (2005). “Wavefront Propagation and Fuzzy Based Autonomous Navigation”. In: *Int. J. of Adv. Robotic Sys.* 2.2, p. 10.

[25] Sakai, Katsuyuki et al. (2000). “What and when: Parallel and convergent processing in motor control”. In: *Journal of Neuroscience* 20.7, pp. 2691–2700.

[26] Sigman, Mariano and Stanislas Dehaene (2008). “Brain mechanisms of serial and parallel processing during dual-task performance”. In: *Journal of Neuroscience* 28.30, pp. 7585–7598.

[27] Davies, Mike (2019b). “Benchmarks for progress in neuromorphic computing”. In: *Nature Machine Intelligence* 1.9, pp. 386–388.

[28] Hart, Peter, Nils Nilsson, and Bertram Raphael (1968). “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

[29] Cormen, Thomas H et al. (1990). *Introduction to Algorithms*. 3rd ed.

[30] Stentz, Anthony (1994). “Optimal and efficient path planning for partially-known environments”. In: *Int. Conf. on Robotics and Automation*, pp. 3310–3317.

[31] Fan, Dong Kai and Ping Shi (2010). “Improvement of Dijkstraa’s algorithm and its application in route planning”. In: *Int. Conf. on Fuzzy Systems and Knowledge Discovery (FSKD)* 4, pp. 1901–1904.

[32] Guruji, Akshay Kumar, Himansh Agarwal, and D.K. Parsediya (2016). “Time-efficient A* Algorithm for Robot Path Planning”. In: *Procedia Technology* 23, pp. 144–149.

[33] Naderi, Kourosh, Joose Rajamäki, and Perttu Hämäläinen (2015). “RT-RRT*: a real-time path planning algorithm based on RRT*”. In: *ACM SIGGRAPH Conf. on Motion in Games* 8, pp. 113–118.

[34] Kuffner, J.J. and S.M. LaValle (2000). “RRT-connect: An efficient approach to single-query path planning”. In: *Int. Conf. on Robotics and Automation* 2, pp. 995–1001.

[35] Ijspeert, A.J., J. Nakanishi, and S. Schaal (2001). "Trajectory formation for imitation with nonlinear dynamical systems". In: *Int. Conf. on Intelligent Robots and Systems*. Vol. 2. IEEE, pp. 752–757.

[36] Ginesi, Michele, Nicola Sansonetto, and Paolo Fiorini (2021). "Overcoming Some Drawbacks of Dynamic Movement Primitives". In: *Robotics and Autonomous Systems* 144, p. 103844.

[37] Hong Qu et al. (2009). "Real-Time Robot Path Planning Based on a Modified Pulse-Coupled Neural Network Model". In: *IEEE Trans. on NN* 20.11, pp. 1724–1739.

[38] Hopfield, J. J. (2010). "Neurodynamics of mental exploration". In: *Proc. of the National Academy of Sciences* 107.4, pp. 1648–1653.

[39] Ponulak, F. and J. Hopfield (2013). "Rapid, parallel path planning by propagating wavefronts of spiking neural activity". In: *Front. in Comp. Neuroscience* 7, p. 98.

[40] Tolman, Edward C. (1948). "Cognitive maps in rats and men". In: *Psychological Review* 55.4, pp. 189–208.

[41] Grieves, Roddy M. and Kate J. Jeffery (2017). "The representation of space in the brain". In: *Behavioural Processes* 135, pp. 113–131.

[42] O'Keefe, John. and Dulcie H. Conway (1978). "Hippocampal place units in the freely moving rat: why they fire where they fire". In: *Experimental brain research* 31.4, pp. 573–590.

[43] Khajeh-Alijani, A., R. Urbanczik, and W. Senn (2015). "Scale-Free Navigational Planning by Neuronal Traveling Waves". In: *PLOS ONE* 10.7. Ed. by William W Lytton, e0127269.

[44] Zennir, M., M. Benmohammed, and R. Boudjadja (2015). "Spike-time dependant plasticity in a spiking neural network for robot path planning". In: *CEUR Workshop Proc.* Vol. 1539, pp. 2–13.

[45] Tanneberg, Daniel, J. Peters, and E. Rueckert (2015). "Spiking neural networks solve robot planning problems". Master thesis.

[46] Tanneberg, Daniel et al. (2016). "Deep Spiking Networks for Model-based Planning in Humanoids". In: *Int. Conf. on Humanoid Robots (HUMANOIDS)*.

[47] Krichmar, Jeffrey L. (2016). "Path planning using a spiking neuron algorithm with axonal delays". In: *ICEC*, pp. 1219–1226.

[48] Hwu, Tiffany, Jeffrey Krichmar, and Xinyun Zou (2017). "A complete neuromorphic solution to outdoor navigation and path planning". In: *Int. Symp. Circuits Syst. (ISCAS)*.

[49] Hwu, Tiffany et al. (2018). "Adaptive robot path planning using a spiking neuron algorithm with axonal delays". In: *IEEE Transactions on Cognitive and Developmental Systems* 10.2, pp. 126–137.

[50] Scharstein, Daniel and Richard Szeliski (2002). "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms". In: *Int. J. of Computer Vision* 47.1-3, pp. 7–42.

[51] Seitz, Steven M. et al. (2006). "A comparison and evaluation of multi-view stereo reconstruction algorithms". In: *Conf. on Computer Vision and Pattern Recognition (CVPR)* 1, pp. 519–526.

[52] Firouzi, Mohsen and Jörg Conradt (2016). “Asynchronous Event-based Cooperative Stereo Matching Using Neuromorphic Silicon Retinas”. In: *Neural Processing Letters* 43.2, pp. 311–326.

[53] Piatkowska, Ewa, Ahmed Nabil Belbachir, and Margrit Gelautz (2013). “Asynchronous stereo vision for event-driven dynamic stereo sensor using an adaptive cooperative approach”. In: *Int. Conf. on Computer Vision*, pp. 45–50.

[54] Osswald, Marc et al. (2017). “A spiking neural network model of 3D perception for event-based neuromorphic stereo vision systems”. In: *Scientific Reports* 7.

[55] Dikov, Georgi et al. (2017). “Spiking Cooperative Stereo-Matching at 2 ms Latency with Neuromorphic Hardware”. In: *Living Machines*, pp. 119–137.

[56] Poggio, Gian F. et al. (1985). “Responses of neurons in visual cortex (V1 and V2) of the alert macaque to dynamic random-dot stereograms”. In: *Vision Research* 25.3, pp. 397–406.

[57] Albada, Sacha J. van et al. (2018). “Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model”. In: *Frontiers in Neuroscience* 12.MAY.

[58] Knight, James C. and Thomas Nowotny (2018). “GPUs Outperform Current HPC and Neuromorphic Solutions in Terms of Speed and Energy When Simulating a Highly-Connected Cortical Model”. In: *Frontiers in Neuroscience* 12, p. 941.

[59] Diamond, Alan, Thomas Nowotny, and Michael Schmuker (2016). “Comparing Neuromorphic Solutions in Action: Implementing a Bio-Inspired Solution to a Benchmark Classification Task on Three Parallel-Computing Platforms”. In: *Frontiers in Neuroscience* 9.JAN, p. 491.

[60] Ostrau, Christoph et al. (2020). “Benchmarking Deep Spiking Neural Networks on Neuromorphic Hardware”. In: *Artificial Neural Networks and Machine Learning*.

[61] DeWolf, Travis, Pawel Jaworski, and Chris Eliasmith (2020). “Nengo and Low-Power AI Hardware for Robust, Embedded Neurorobotics”. In: *Frontiers in Neurorobotics* 14.

[62] Yan, Yexin et al. (2021). “Comparing Loihi with a SpiNNaker 2 prototype on low-latency keyword spotting and adaptive robotic control”. In: *Neuromorphic Computing and Engineering*.

[63] Arbib, Michael (1995). *The Handbook of Brain Theory and Neural Networks*.

[64] Kandel, Eric R., James H. Schwartz, and Thomas M. Jessell (1999). *Principles of neural science*. 4th ed. New York: Elsevier, pp. 317–330.

[65] Dröscher, Ariane (1998). “Camillo Golgi and the discovery of the Golgi apparatus”. In: *Histochemistry and Cell Biology 1998 109:5* 109.5, pp. 425–430.

[66] Cajal, Santiago Ramón y (1967). *Nobel Lectures: Physiology or Medicine*.

[67] Gray, Henry and Warren H Lewis (1918). *Anatomy of the Human Body*. 20th ed. Philadelphia.

[68] Pakkenberg, B and H.J. Gundersen (1997). "Neocortical neuron number in humans: effect of sex and age". In: *J. of comparative neurology* 384.2, pp. 312–20.

[69] Gerstner, Wulfram et al. (2014). *Neuronal Dynamics - From Single Neurons to Networks and Models of Cognition*. Cambridge University Press.

[70] Dharani, Krishnagopal (2015). "Physiology of the Neuron". In: *The Biology of Thought*, pp. 31–52.

[71] Dayan, Peter. and L. F. Abbott (2001). *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. Massachusetts Institute of Technology Press, p. 460.

[72] Fiala, John C., Josef Spacek, and Kristen M. Harris (2012). "Dendrite structure". In: *Dendrites*.

[73] Khan, Salman (2010). *The membrane potential - How the resting membrane potential is established in a neuron.* Tech. rep.

[74] Herculano-Houzel, Suzana (2009). "The human brain in numbers: A linearly scaled-up primate brain". In: *Frontiers in Human Neuroscience* 3, p. 31.

[75] Voglis, Giannis and Nektarios Tavernarakis (2006). "The role of synaptic ion channels in synaptic plasticity". In: *EMBO Reports* 7.11, p. 1104.

[76] Citri, Ami and R. Malenka (2007). "Synaptic Plasticity: Multiple Forms, Functions, and Mechanisms". In: *Neuropsychopharmacology* 33.1, pp. 18–41.

[77] Zucker, Robert S. and Wade G. Regehr (2002). "Short-term synaptic plasticity". In: *Annual review of physiology* 64, pp. 355–405.

[78] Hennig, Matthias H. (2013). "Theoretical models of synaptic short term plasticity". In: *Frontiers in Computational Neuroscience* 7.

[79] Morris, Richard G.M. (2003). "Long-term potentiation and memory". In: *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 358.1432, pp. 643–647.

[80] Wang, Yun et al. (2006). "Heterogeneity in the pyramidal network of the medial prefrontal cortex". In: *Nature neuroscience* 9.4, pp. 534–542.

[81] Abbott, L. F. and Wade G. Regehr (2004). "Synaptic computation". In: *Nature* 431.7010, pp. 796–803.

[82] Hebb, D. (1949). *The organization of behavior: A neuropsychological theory*. New York: L. Erlbaum Associates.

[83] Bi, G. Q. and M. M. Poo (2001). "Synaptic Modification by Correlated Activity: Hebb's Postulate Revisited". In: *Ann. Rev. Neurosci.* 24, pp. 139–166.

[84] Baldi, Pierre and Peter Sadowski (2016). "A theory of local learning, the learning channel, and the optimality of backpropagation". In: *Neural Networks* 83, pp. 51–74.

[85] Stevens, C F (1996). "Strengths and weaknesses in memory". In: *Nature* 381, pp. 471–472.

[86] Carr, C. E. and M. Konishi (1990). "A circuit for detection of interaural time differences in the brain stem of the barn owl". In: *Journal of Neuroscience* 10, pp. 3227–3246.

[87] Gütig, R. et al. (2003). "Learning input correlations through nonlinear temporally asymmetric Hebbian plasticity". In: *Journal of Neuroscience* 23.9, pp. 3697–3714.

[88] D J Linden (1999). "The return of the spike: postsynaptic action potentials and the induction of LTP and LTD". In: *Neuron*.

[89] Frégnac, Yves et al. (2010). "A re-examination of Hebbian-covariance rules and spike timing-dependent plasticity in cat visual cortex in vivo". In: *Frontiers in Synaptic Neuroscience* 0.DEC, p. 147.

[90] Lisman, John and Nelson Spruston (2010). "Questions about STDP as a general model of synaptic plasticity". In: *Frontiers in Synaptic Neuroscience* 0.OCT, pp. 1–5.

[91] Cottrell, Marie et al. (2016). "Theoretical and applied aspects of the self-organizing maps". In: *WSOM* 11, pp. 3–26.

[92] Banzhaf, Wolfgang (2009). "Self-organizing Systems". In: *Encyclopedia of Complexity and Systems Science*. Ed. by R. A. Meyers, pp. 8040–8050.

[93] Kohonen, Teuvo (1989). "Self-Organization and Associative Memory". In: Springer Series in Information Sciences 8.

[94] Willshaw, D. J. and C. Von Der Malsburg (1976). "How Patterned Neural Connections Can Be Set Up by Self-Organization on JSTOR". In: *Proc. R. Soc. Lond. B Series B, Biological Sciences* 194.1117, pp. 431–445.

[95] Kohonen, Teuvo, Kohonen, and Teuvo (1986). "Representation Of Sensory Information In Self-Organizing Feature Maps, And Relation Of These Maps To Distributed Memory Networks". In: *SPIE* 634, pp. 248–259.

[96] Rumbell, Timothy, Susan L. Denham, and Thomas Wennekers (2014). "A spiking self-organizing map combining STDP, oscillations, and continuous learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.5, pp. 894–907.

[97] Van Hulle, Marc M (2012). *Self-Organizing Maps*. Tech. rep.

[98] T, Kohonen (1990). "The self-organizing map". In: *Proceedings of the IEEE* 78.9, pp. 1464–1480.

[99] Schott, G. D. (1993). "Penfield's homunculus: A note on cerebral cartography". In: *Journal of Neurology Neurosurgery and Psychiatry* 56.4, pp. 329–333.

[100] Merzenich, M. M., P. L. Knight, and G. L. Roth (1975). "Representation of cochlea within primary auditory cortex in the cat". In: *Journal of Experimental Psychology* 38.2, pp. 231–249.

[101] Chen, X et al. (2011). "A gustotopic map of taste qualities in the mammalian brain". In: *Science* 333, pp. 1262–1266.

[102] Woolsey, Thomas A., Carol Welker, and Richard H. Schwartz (1975). "Comparative anatomical studies of the Sml face cortex with special reference to the occurrence of "barrels" in layer IV". In: *Journal of Comparative Neurology* 164.1, pp. 79–94.

[103] Friedman, Robert M., Li Min Chen, and Anna Wang Roe (2004). "Modality maps within primate somatosensory cortex". In: *Proc. of the National Academy of Sciences of the USA* 101.34, pp. 12724–12729.

[104] Malsburg, Chr von der (1973). "Self-organization of orientation sensitive cells in the striate cortex". In: *Kybernetik* 14.2, pp. 85–100.

[105] Millodot, Michel (2009). *Dictionary of optometry and vision science*, p. 388.

[106] Posch, Christoph et al. (2014). "Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output". In: *Proceedings of the IEEE* 102.10, pp. 1470–1484.

[107] Boahen, K. (1996). "Retinomorphic vision systems". In: *Microelectronics for Neural Networks*, pp. 2–14.

[108] Rodieck, R. W. (1998). *The First Steps in Seeing*.

[109] Goldstein, E. Bruce. (2015). *Wahrnehmungspsychologie : der Grundkurs*.

[110] Nassi, Jonathan J. and Edward M. Callaway (2009). "Parallel processing strategies of the primate visual system". In: *Nature Reviews Neuroscience* 10.5, pp. 360–372.

[111] Sincich, Lawrence C. and Jonathan C. Horton (2005). "The circuitry of V1 and V2: Integration of color, form, and motion". In: *Annual Review of Neuroscience* 28, pp. 303–326.

[112] Boahen, Kwabena A. (1998). "Communicating Neuronal Ensembles between Neuromorphic Chips". In: *Neuromorphic Systems Engineering*, pp. 229–259.

[113] Boahen, Kwabena (2005). "Neuromorphic Microchips". In: *Scientific American* 16.3s, pp. 20–27.

[114] Julesz, Bela (1964). "Binocular Depth Perception without Familiarity Cues". In: *Science*.

[115] Walk, Richard D. (1966). "The Development of Depth Perception in Animals and Human Infants". In: *Monographs of the Society for Research in Child Development* 31.5, p. 82.

[116] Timney, Brian (1985). "Visual Experience and the Development of Depth Perception". In: *Brain Mechanisms and Spatial Vision*, pp. 147–174.

[117] Hauser, L. (2016). *Depth perception - lecture notes*. Tech. rep. University of Calgary.

[118] Richards, Whitman and John F. Miller (1969). "Convergence as a cue to depth". In: *Perception & Psychophysics* 5.5, pp. 317–320.

[119] Ciuffreda, Kenneth J. (2006). "Accommodation, the Pupil, and Presbyopia". In: *Borish's Clinical Refraction*, pp. 93–144.

[120] Ganong, William F. (1971). *Medizinische Physiologie*. Springer Berlin Heidelberg.

[121] Cutting, J. (1997). "High-performance Computing and Human Vision". In: *Behavior Research Methods, Instruments & Computers* 29, pp. 27–36.

[122] Rose, D. (1980). "The binocular: monocular sensitivity ratio for movement detection varies with temporal frequency". In: *Perception* 9.5, pp. 577–580.

[123] Adachi-Usami, E. and D. Lehmann (1983). "Monocular and binocular evoked average potential field topography: Upper and lower hemiretinal stimuli". In: *Experimental Brain Research* 50.2, pp. 341–346.

[124] Marr, D. and T. Poggio (1976). "Cooperative computation of stereo disparity". In: *Science* 194.4262, pp. 283–287.

[125] Julesz, Bela (1960). "Binocular depth perception of computer-generated patterns". In: *Bell System Technical Journal* 39, pp. 1125–1162.

[126] Granrud, Carl E., Albert Yonas, and Linda Pettersen (1984). "A comparison of monocular and binocular depth perception in 5- and 7-month-old infants". In: *Journal of Experimental Child Psychology* 38.1, pp. 19–32.

[127] Papathomas, Thomas V., Kazunori Morikawa, and Nicholas Wade (2019). "Bela julesz in depth". In: *Vision (Switzerland)* 3.2.

[128] Wheatstone, C. (1838). "Contributions to the physiology of vision. —Part the first. On some remarkable, and hitherto unobserved, phenomena of binocular vision". In: *Philosophical Transactions of the Royal Society of London* 128, pp. 371–394.

[129] Julesz, Bela. (1971). *Foundations of cyclopean perception*. The University of Chicago Press, p. 406.

[130] Solstad, Trygve, Edvard I. Moser, and Gaute T. Einevoll (2006). "From grid cells to place cells: A mathematical model". In: *Hippocampus* 16.12, pp. 1026–1031.

[131] Eichenbaum, H. et al. (1999). *The Hippocampus , Memory , Review and Place Cells : Is It Spatial Memory or a Memory Space ?*

[132] O'Keefe, J. and J. Dostrovsky (1971). "The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat". In: *Brain Research* 34.1, pp. 171–175.

[133] Muller, R. U., J. L. Kubie, and J. B. Ranck (1987). "Spatial firing patterns of hippocampal complex-spike cells in a fixed environment". In: *Journal of Neuroscience* 7, pp. 1935–1950.

[134] Wilson, Matthew A. and Bruce L. McNaughton (1993). "Dynamics of the hippocampal ensemble code for space". In: *Science* 261.5124, pp. 1055–1058.

[135] Ekstrom, Arne D. et al. (2003). "Cellular networks underlying human spatial navigation". In: *Nature* 425.6954, pp. 184–187.

[136] Alme, Charlotte B. et al. (2014). "Place cells in the hippocampus: Eleven maps for eleven rooms". In: *Proc. of the National Academy of Sciences* 111.52, pp. 18428–18435.

[137] Fyhn, M. et al. (2004). "Spatial Representation in the Entorhinal Cortex". In: *Science* 305.5688, pp. 1258–1264.

[138] Hafting, Torkel et al. (2005). "Microstructure of a spatial map in the entorhinal cortex". In: *Nature* 436.7052, pp. 801–806.

[139] Taube, J. S., R. U. Muller, and J. B. Ranck (1990). "Head-direction cells recorded from the postsubiculum in freely moving rats." In: *Journal of Neuroscience* 10.2, pp. 420–435.

[140] Lapicque, Louis (1907). "Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation". In: *J. Physiol. Pathol. Gen.*

[141] Abbott, L F (1999). "Lapicque's introduction of the integrate-and-fire model neuron (1907)". In: *Brain Research Bulletin* 50.6, pp. 303–304.

[142] Brunel, N. and M. Van Rossum (2007). "Lapicque's 1907 paper: From frogs to integrate-and-fire". In: *Biological Cybernetics* 97.5-6, pp. 337–339.

[143] Burkitt, A. N. (2006). "A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input". In: *Biological Cybernetics* 95.1, pp. 1–19.

[144] Gerstner, Wulfram (1995). "Time structure of the activity in neural network models". In: *Physical Review E* 51.1, p. 738.

[145] Hodgkin, A. L. and A. F. Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *J. of Physiology* 117.4, pp. 500–544.

[146] Izhikevich, Eugene M (2003). "Simple Model of Spiking Neurons". In: *Trans. on ANN* 14.6.

[147] Schmidhuber, J. (2015). "Deep learning in neural networks: An overview". In: *Neural Networks* 61, pp. 85–117.

[148] Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning Internal Representations by Error Propagation*.

[149] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature 1986 323:6088* 323.6088, pp. 533–536.

[150] Bartunov, Sergey et al. (2018). "Assessing the scalability of biologically-motivated deep learning algorithms and architectures". In: *Neural Information Processing Systems ((NeurIPS)* 32, pp. 9368–9378.

[151] Bengio, Yoshua et al. (2015). "Towards Biologically Plausible Deep Learning". In: *arXiv:1502.04156*.

[152] Bohte, S., J. Kok, and H. La Poutré (2002). "Error-backpropagation in temporally encoded networks of spiking neurons". In: *Neurocomputing* 48, pp. 17–37.

[153] Neftci, Emre et al. (2016). "Neuromorphic Deep Learning Machines". In: *Frontiers in Neuroscience* 11.JUN, p. 324.

[154] Zenke, Friedemann and Surya Ganguli (2018). "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks". In: *Neural Computation* 30.6, pp. 1514–1541.

[155] Kempter, Richard, Wulfram Gerstner, and J. Leo van Hemmen (1999). "Hebbian learning and spiking neurons". In: *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics* 59.4, pp. 4498–4514.

[156] Song, Sen, Kenneth Miller, and L. Abbott (2000). "Competitive Hebbian Learning Through Spike-Timing-Dependent Synaptic Plasticity". In: *Nature Neuroscience* 3, pp. 919–26.

[157] Koch, Giacomo et al. (2013). "Hebbian and anti-Hebbian spike-timing-dependent plasticity of human cortico-cortical connections". In: *Journal of Neuroscience* 33.23, pp. 9725–9733.

[158] Walter, Florian, Florian Röhrbein, and Alois Knoll (2015). "Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks". In: *Neural Networks* 72, pp. 152–167.

[159] Tavanaei, Amirhossein et al. (2019). "Deep Learning in Spiking Neural Networks". In: *Neural Networks* 111, pp. 47–63.

[160] Blundell, Inga et al. (2018). "Code Generation in Computational Neuroscience: A Review of Tools and Techniques". In: *Frontiers in Neuroinformatics* 12, p. 68.

[161] Brette, Romain and Dan F.M. Goodman (2012). "Simulating spiking neural networks on GPU". In: *Network: Computation in Neural Systems* 23.4, pp. 167–182.
[162] Mead, Carver (1990). "Neuromorphic Electronic Systems". In: *Proceedings of the IEEE* 78.10, pp. 1629–1636.
[163] Indiveri, Giacomo. *Introducing "Neuromorphic Computing and Engineering"*. Tech. rep.
[164] Frenkel, Charlotte et al. (2021). *Bottom-Up and Top-Down Neural Processing Systems Design: Neuromorphic Intelligence as the Convergence of Natural and Artificial Intelligence*. Tech. rep.
[165] Bains, Sunny (2020). "The Promise and Pitfalls of Neuromorphic Computers". In: *EE Times*.
[166] Joubert, A. et al. (2012). "Hardware spiking neurons design: Analog or digital?" In: *Int. Joint Conf. on Neural Networks*.
[167] Renaud, Sylvie et al. (2007). "Neuromimetic ICs with analog cores: An alternative for simulating spiking neural networks". In: *Int. Symp. Circuits Syst. (ISCAS)*, pp. 3355–3358.
[168] Benjamin, Ben Varkey et al. (2014). "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations". In: *Proceedings of the IEEE* 102.5, pp. 699–716.
[169] Müller, Eric et al. (2020a). "Extending BrainScaleS OS for BrainScaleS-2". In: *arXiv:2003.13750*.
[170] Schemmel, Johannes et al. (2022). "Accelerated Analog Neuromorphic Computing". In: *Analog Circuits for Machine Learning, Current/Voltage/Temperature Sensors, and High-speed Communication*, pp. 83–102.
[171] Müller, Eric et al. (2020b). "The Operating System of the Neuromorphic BrainScaleS-1 System". In: *Neurocomputing* 501, pp. 790 –810.
[172] Merolla, P. A. et al. (2014). "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197, pp. 668–673.
[173] Schuman, Catherine D et al. (2017). "A Survey of Neuromorphic Computing and Neural Networks in Hardware". In: *ArXiv*.
[174] Shrestha, Amar et al. (2022). "A survey on neuromorphic computing: Models and hardware". In: *IEEE Circuits and Systems Magazine* 22.2, pp. 6–35.
[175] Oltra-Oltra, J. et al. (2021). "Hardware-software co-design for efficient and scalable real-time emulation of SNNs on the edge". In: *Int. Symp. Circuits Syst. (ISCAS)*, pp. 1–5.
[176] Astudillo, C. and B. Oommen (2014). "Topology-oriented self-organizing maps: A survey". In: *Pattern Analysis and Applications* 17.2, pp. 223–248.
[177] Miljkovic, Dubravko (2017). "Brief review of self-organizing maps". In: *MIPRO*, pp. 1061–1066.
[178] Kohonen, Teuvo (1982). "Self-organized formation of topologically correct feature maps". In: *Biological Cybernetics* 43.1, pp. 59–69.
[179] — (1997). *Self-Organizing Maps*.
[180] — (1988). "The "Neural" Phonetic Typewriter". In: *Computer* 21.3.

[181] Liu, Yuan-Chao, Ming Liu, and Xiao-Long Wang (2012). "Application of Self-Organizing Maps in Text Clustering: A Review". In: *Applications of Self-Organizing Maps*.

[182] Barreto, G. and A. Araújo (2001). "Time in self-organizing maps: An overview of models". In: *Int. J. of Computer Research* 10.2, pp. 139–179.

[183] Himmelblau, David M (2000). "Applications of Artificial Neural Networks in Chemical Engineering". In: *Korean J. Chem. Eng* 17.4, pp. 373–392.

[184] Dondi, Daniele, Armando Buttafava, and Angelo Albini (2011). "Application of Self-Organizing Maps in Chemistry. The Case of Phenyl Cations". In: *Self Organizing Maps - Applications and Novel Algorithm Design*. Ed. by Josphat Igadwa Mwasiagi. InTech. Chap. 21.

[185] Aldrich, C., D. Moolman, and J. van Deventer (1995). "Monitoring and control of hydrometallurgical processes with self-organizing and adaptive neural net systems". In: *Computers and Chemical Engineering* 19, pp. 803–808.

[186] Kaski, Samuel, J. Kangas, and T. Kohonen (1998). "Bibliography of Self-Organizing Map (SOM) Papers: 1981-1997". In: *Neural Computing Surveys* 1.3 & 4, pp. 1–176.

[187] Oja, Merja, Samuel Kaski, and Tuevo Kohonen (2003). "Bibliography of Self-Organizing Map (SOM) Papers: 1998-2001 Addendum". In: *CiteSeer* 3, pp. 1–156.

[188] Polla, Matti, Timo Honkela, and Tuevo Kohonen (2006). "Bibliography of self-organizing map (SOM) papers: 2002–2005". In: *TKK Reports in Information and Computer Science*.

[189] Reimann, D. and H. Haken (1994). "Stereo vision by self-organization". In: *Biological Cybernetics 1994 71:1* 71.1, pp. 17–26.

[190] Pajares, G., J. M. Cruz, and J. Aranda (1998). "Stereo matching based on the self-organizing feature-mapping algorithm". In: *Pattern Recognition Letters* 19.3-4, pp. 319–330.

[191] Guan, Haiying, Rogerio S. Feris, and Matthew Turk (2006). "The Isometric Self-Organizing Map for 3D hand pose estimation". In: *Proceedings of the 7th International Conference on Automatic Face and Gesture Recognition*, pp. 263–268.

[192] Sajó, Levente, Miklós Hoffmann, and Attila Fazekas (2009). "A 3D head model from stereo images by a self-organizing neural network". In: *Journal for Geometry and Graphics* 13.2, pp. 1–12.

[193] Yusob, B., S. Shamsuddin, and H. Hamed (2013). "Spiking Self-organizing Maps for Classification Problem". In: *Procedia Technology* 11, pp. 57–64.

[194] Hazan, Hananel et al. (2018). "Unsupervised Learning with Self-Organizing Spiking Neural Networks". In: *Proc. of the Int. Joint Conf. on Neural Networks* 2018-July.

[195] Asan, Umut and Secil Ercan (2012). "An Introduction to Self-Organizing Maps". In: *Computational Intelligence Systems in Industrial Engineering: with Recent Theory and Applications*. Ed. by C. Kahraman. Atlantis Press. Chap. 14, pp. 295–315.

[196] Fort, Jean-Claude and Gilles Pagès (1994). "About the convergence of the generalized Kohonen algorithm". In: *ICANN*, pp. 318–321.
[197] Cottrell, Marie et al. (2018). "Self-Organizing Maps, theory and applications". In: *Revista de Investigacion Operacional* 39.1, pp. 1–22.
[198] Hamel, Lutz (2018). "Vsom: Efficient, stochastic self-organizing map training". In: *Advances in Intelligent Systems and Computing* 869, pp. 805–821.
[199] Kohonen, Tuevo (1999). "Comparison of SOM point densities based on different criteria". In: *Neural computation* 11.8, pp. 2081–2095.
[200] Koikkalainen, Pasi and Erkki Oja (1990). "Self-organizing hierarchical feature maps". In: *IJCNN. International Joint Conference on Neural Networks*, pp. 279–284.
[201] Martinetz, T. and K. Schulten (1991). "A Neural-Gas Network Learns Topologies". In: *Artificial Neural Networks*, pp. 397–402.
[202] Martinetz, Thomas and Klaus Schulten (1994). "Topology representing networks". In: *Neural Networks* 7.3, pp. 507–522.
[203] Fritzke, B. (1995). "A Growing Neural Gas Network Learns Topologies". In: *Adv. in Neural Information Processing Systems*.
[204] Hammer, B. et al. (2005). "Self organizing maps for time series". In: *WSOM*, pp. 115–122.
[205] Andreakis, A., N. Hoyningen-Huene, and M. Beetz (2009). "Incremental unsupervised time series analysis using merge growing neural gas". In: *Lecture Notes in Computer Science* 5629, pp. 10–18.
[206] Vergara, Jorge R., Pablo A. Estévez, and Álvaro Serrano (2016). "Segment growing neural gas for nonlinear time series analysis". In: *Adv. Intell. Syst. Comput.* 428, pp. 107–117.
[207] Vergara, Jorge R. and Pablo A. Estévez (2017). "A strategy for time series prediction using segment growing neural gas". In: *WSOM*.
[208] Vathy-Fogarassy, Agnes et al. (2007). "Visualization of topology representing networks". In: *Lecture Notes in Computer Science* 4881 LNCS, pp. 557–566.
[209] Yin, Hujun (2002). "ViSOM-a novel method for multivariate data projection and structure visualization". In: *IEEE Transactions on Neural Networks* 13.1, pp. 237–243.
[210] Fiannaca, Antonino et al. (2007). "Improved SOM learning using simulated annealing". In: *Lecture Notes in Computer Science* 4668, pp. 279–288.
[211] Berglund, Erik and Joaquin Sitte (2006). "The parameterless self-organizing map algorithm". In: *IEEE Transactions on Neural Networks* 17.2, pp. 305–316.
[212] Karin Haese (2001). "Auto-SOM: Recursive Parameter Estimation for Guidance of Self-Organizing Feature Maps". In: *Neural Computation* 13, pp. 595–619.
[213] Si, J., S. Lin, and M. Vuong (2000). *Dynamic topology representing networks*.
[214] Muhammed, Hamed Hamid (2003). "Unsupervised fuzzy clustering and image segmentation using weighted neural networks". In: *Int. Conf. on Image Analysis and Processing* 12, pp. 308–313.

[215] Costa, José Alfredo F. and Ricardo S. Oliveira (2007). "Cluster analysis using growing neural gas and graph partitioning". In: *Int. Conf. on Neural Networks*, pp. 3051–3056.

[216] Canales, Fernando and Max Chacón (2007). "Modification of the Growing Neural Gas Algorithm for Cluster Analysis". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4756 LNCS, pp. 684–693.

[217] Fritzke, Bernd (1997). "Unsupervised Ontogenic Networks". In: *Handbook of Neural Computation* C2.4.

[218] Vojáček, Lukáš et al. (2016). "Optimization of Combining of Self Organizing Maps and Growing Neural Gas". In: *Lecture Notes in Computer Science*, pp. 277–286.

[219] Prudent, Yann and Abdellatif Ennaji (2005). "An incremental growing neural gas learns topologies". In: *Int. J. Conf. on Neural Networks* 2, pp. 1211–1216.

[220] Martinetz, Thomas (1993). "Competitive Hebbian Learning Rule Forms Perfectly Topology Preserving Maps". In: *ICANN*, pp. 427–434.

[221] Simon, Geoffroy et al. (2003). "Double SOM for long-term time series prediction". In: *WSOM*, pp. 35–40.

[222] Vesanto, Juha (1997). "Using the SOM and Local Models in Time Series Prediction". In: *WSOM*, pp. 209–214.

[223] Somervuo, Panu J. (2004). "Online algorithm for the self-organizing map of symbol strings". In: *Neural Networks* 17.8-9, pp. 1231–1239.

[224] Bishop, Christopher M., Geoffrey E. Hinton, and Iain G. D. Strachan (1997). "GTM through time — Aston Research Explorer". In: *ICANN*, pp. 111–116.

[225] Tino, Peter et al. (2004). "A Generative Probabilistic Approach to Visualizing Sets of Symbolic Sequences". In: *Int. conf. on Knowledge discovery and data mining* 4, pp. 701–706.

[226] Euliano, Neil R. and Jose C. Principe (1999). "A Spatio-Temporal Memory Based on SOMs with Activity Diffusion". In: *Kohonen Maps*. Elsevier Science B.V., pp. 253–265.

[227] Schulz, Reiner and James A. Reggia (2004). "Temporally Asymmetric Learning Supports Sequence Processing in Multi-Winner Self-Organizing Maps". In: *Neural Computation* 16.3, pp. 535–561.

[228] Wiemer, Jan C (2003). "The Time-Organized Map Algorithm: Extending the Self-Organizing Map to Spatiotemporal Signals". In: *Neural Computation* 15, pp. 1143–1171.

[229] Chappell, Geoffrey J. and John G. Taylor (1993). "The temporal Kohønen map". In: *Neural Networks* 6.3, pp. 441–445.

[230] Koskela, Timo et al. (1998). "Temporal sequence processing using recurrent SOM". In: *Int. Conf. on Knowledge-Based Intelligent Electronic Systems (KES)* 2, pp. 290–297.

[231] Voegtlin, Thomas and Peter F. Dominey (2001). "Recursive Self-Organizing Maps". In: *Adv. in SOM*. Springer London, pp. 210–215.

[232] Hagenbuchner, Markus, Alessandro Sperduti, and Ah Chung Tsoi (2003). "A self-organizing map for adaptive processing of structured data". In: *Trans. on ANN* 14.3, pp. 491–505.

[233] Strickert, Marc and Barbara Hammer (2005). "Merge SOM for temporal data". In: *Neurocomputing* 64, pp. 39–71.

[234] Estévez, Pablo A. and Rodrigo Hernández (2009). "Gamma SOM for temporal sequence processing". In: *Lect. Notes Comput. Sci.* 5629, pp. 63–71.

[235] Estévez, Pablo A. and Jorge R. Vergara (2013). "Nonlinear Time Series Analysis by Using Gamma Growing Neural Gas". In: *Adv. Intell. Syst. Comput.* 198, pp. 205–214.

[236] Graham, James and Janusz A. Starzyk (2008). "A hybrid self-organizing neural gas based network". In: *Proc. of the Int. J. Conf. on NN*, pp. 3806–3813.

[237] Forest, Florent et al. (2020). "A Survey and Implementation of Performance Metrics for Self-Organized Maps". In: *CoRR*.

[238] Cabanes, Guénaël and Younès Bennani (2010). "Learning topological constraints in Self-Organizing Map". In: *Int. Conf. on Neural Information Processing* 17, pp. 367–374.

[239] Goodhill, Geoffrey and Terrence Sejnowski (1996). "Quantifying neighbourhood preservation in topographic mappings". In: *Joint Sympostium on Neural Computation* 6.3, pp. 61–69.

[240] Furmonas, Justas, John Liobe, and Vaidotas Barzdenas (2022). "Analytical Review of Event-Based Camera Depth Estimation Methods and Systems". In: *Sensors* 22.3, p. 1201.

[241] Etienne-Cummings, R. and J. Van Der Spiegel (1996). "Neuromorphic vision sensors". In: *Sensors and Actuators* 56.1-2, pp. 19–29.

[242] Mahowald, M. and C. Mead (1991). "The silicon retina". In: *Scientific American* 264.5, pp. 76–83.

[243] Mahowald, Misha (1994). *An Analog VLSI System for Stereoscopic Vision*. Springer US.

[244] Lichtsteiner, Patrick, Christoph Posch, and Tobi Delbruck (2008). "A 128 × 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor". In: *IEEE Journal of Solid-State Circuits* 43.2, pp. 566–576.

[245] Zaghloul, Kareem A. and Kwabena Boahen (2006). "A silicon retina that reproduces signals in the optic nerve". In: *Journal of Neural Engineering* 3.4.

[246] Ruedi, Pierre François et al. (2003). "A 128 × 128 pixel 120dB dynamic range vision sensor chip for image contrast and orientation extraction". In: *Int. Solid-State Circuits Conf.*

[247] Mallik, Udayan et al. (2005). "Temporal change threshold detection imager". In: *Int. Solid-State Circuits Conf.* 48, pp. 298–299.

[248] Lazzaro, John et al. (1993). "Silicon Auditory Processors as Computer Peripherals". In: *IEEE Transactions on Neural Networks* 4.3, pp. 523 –528.

[249] Berner, R. et al. (2013). "A 240x180 120dB 10mW 12us-latency sparse output vision sensor for mobile applications". In: *Symposium on VLSI Circuits*.

[250] Brandli, Christian et al. (2014). "A 240 × 180 130 dB 3 μs latency global shutter spatiotemporal vision sensor". In: *IEEE Journal of Solid-State Circuits* 49.10, pp. 2333–2341.

[251] Posch, Christoph, Daniel Matolin, and Rainer Wohlgenannt (2011). "A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS". In: *IEEE Journal of Solid-State Circuits* 46.1, pp. 259–275.

[252] *Prophesee Evaluation Kits* (2020).

[253] Suh, Yunjae et al. (2020). "A 1280x960 dynamic vision sensor with a 4.95-μm pixel pitch and motion artifact minimization". In: *Int. Symp. Circuits Syst. (ISCAS)*.

[254] IniVation (2020). *Understanding the Performance of Neuromorphic Event-based Vision Sensors*.

[255] Delbruck, T., V. Villanueva, and L. Longinotti (2014). "Integration of dynamic vision sensor with inertial measurement unit for electronically stabilized event-based vision". In: *Int. Symp. Circuits Syst. (ISCAS)*, pp. 2636–2639.

[256] Berner, Raphael (2006). "Highspeed USB2.0 AER Interfaces". doctoral thesis. ETH Zurich.

[257] Fossum, Eric (1997). "CMOS lmagc Sensors: Electronic Camera on A Chip". In: *IEEE Transactions on Electron Devices*.

[258] Ieng, Sio-Hoi et al. (2018). "Neuromorphic Event-Based Generalized Time-Based Stereovision". In: *Frontiers in Neuroscience* 12, p. 442.

[259] Orchard, Garrick et al. (2014). "Accelerated frame-free time-encoded multi-step imaging". In: *Int. Symp. Circuits Syst. (ISCAS)*, pp. 2644–2647.

[260] Schraml, S., P. Schön, and N. Milosevic (2007). "Smartcam for real-time stereo vision - address-event based embedded system." In: *Int. Conf. on Computer Vision Theory and Applications* 2.

[261] Kogler, Jürgen, Christoph Sulzbachner, and Wilfried Kubinger (2009). "Bio-inspired stereo vision system with silicon retina imagers". In: *Lecture Notes in Computer Science* 5815, pp. 174–183.

[262] Belbachir, A. N. et al. (2012). "CARE: A dynamic stereo vision sensor system for fall detection". In: *Int. Symp. Circuits Syst. (ISCAS)*, pp. 731–734.

[263] Dominguez-Morales, M. Cerezuela-Escudero, E. et al. (2011). *Image Matching Algorithms in Stereo Vision using Address-event-representation - A Theoretical Study and Evaluation of the Different Algorithms*.

[264] Kogler, Jürgen, Martin Humenberger, and Christoph Sulzbachner (2011). "Event-Based Stereo Matching Approaches for Frameless Address Event Stereo Data". In: *Advances in Visual Computing* 7.

[265] Kogler, Juergen et al. (2011). "Address-Event Based Stereo Vision with Bio-Inspired Silicon Retina Imagers". In: *Advances in Theory and Applications of Stereo Vision*.

[266] Ye, Chengxi et al. (2018). "Unsupervised Learning of Dense Optical Flow, Depth and Egomotion from Sparse Event Data". In: *Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 5831–5838.

[267] Maqueda, Ana I. et al. (2018). "Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars". In: *Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 5419–5427.

[268] Tulyakov, Stepan et al. (2019). "Learning an Event Sequence Embedding for Dense Event-Based Deep Stereo". In: *Int. Conf. on Computer Vision*, pp. 1527–1537.

[269] Nguyen, Anh et al. (2019). "Real-time 6DOF pose relocalization for event cameras with stacked spatial LSTM networks". In: *Workshop in Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 1638–1645.

[270] Moeys, Diederik Paul et al. (2016). "Steering a predator robot using a mixed frame/event-driven convolutional neural network". In: *Int. Conf. on Event-Based Control, Communication, and Signal Processing* 2.

[271] Wang, Lin et al. (2019). "Event-based high dynamic range image and very high frame rate video generation using conditional generative adversarial networks". In: *Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 10073–10082.

[272] Benosman, Ryad et al. (2011). "Asynchronous event-based Hebbian epipolar geometry". In: *Neural Networks* 22.11, pp. 1723–1734.

[273] Rogister, Paul et al. (2012). "Asynchronous event-based binocular stereo matching". In: *Neural Networks and Learning Systems* 23.2, pp. 347–353.

[274] Carneiro, João et al. (2013). "Event-based 3D reconstruction from neuromorphic retinas". In: *Neural Networks* 45, pp. 27–38.

[275] Benosman, Ryad et al. (2014). "Event-based visual flow". In: *Neural Networks and Learning Systems* 25.2, pp. 407–417.

[276] Schraml, Stephan, Ahmed Nabil Belbachir, and Horst Bischof (2015). "Event-driven stereo matching for real-time 3D panoramic vision". In: *Conf. on Computer Vision and Pattern Recognition (CVPR)* 07-12-June, pp. 466–474.

[277] Matsuda, Nathan, Oliver Cossairt, and Mohit Gupta (2015). "MC3D: Motion Contrast 3D Scanning". In: *Int. Conf. on Computational Photography (ICCP)*.

[278] Martel, Julien N P et al. (2018). "An Active Approach to Solving the Stereo Matching Problem using Event-Based Sensors". In: *Int. Symp. Circuits Syst. (ISCAS)*.

[279] Rebecq, Henri et al. (2018). "EMVS: Event-Based Multi-View Stereo—3D Reconstruction with an Event Camera in Real-Time". In: *Int. J. of Computer Vision* 126.12, pp. 1394–1414.

[280] Collins, Robert T (1996). "A Space-Sweep Approach t o T rue Multi-Image Matching". In: *Conf. on Computer Vision and Pattern Recognition*, pp. 358–363.

[281] Kim, Hanme, Stefan Leutenegger, and Andrew J. Davison (2016). "Real-time 3D reconstruction and 6-DoF tracking with an event camera". In: *Lecture Notes in Computer Science* 9910 LNCS, pp. 349–364.

[282] Gallego, Guillermo, Henri Rebecq, and Davide Scaramuzza (2018). "A Unifying Contrast Maximization Framework for Event Cameras, with Applications to Motion, Depth, and Optical Flow Estimation". In: *Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 3867–3876.

[283] Haessig, Germain et al. (2019). "A Spiking Neural Network Model of Depth from Defocus for Event-based Neuromorphic Vision". In: *Scientific Reports* 9.1.

[284] Marr, D. and T. Poggio (1977). "A Theory of Human Stereo Vision". In: *Tech. rep., Cambridge*.

[285] — (1979). "A computational theory of human stereo vision". In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 204.1156, pp. 301–328.

[286] Mahowald, Misha (1992). *VLSI analogs of neuronal visual processing : a synthesis of form and function*. Pasadena.

[287] Piatkowska, Ewa et al. (2017). "Improved Cooperative Stereo Matching for Dynamic Vision Sensors with Ground Truth Evaluation". In: *Workshop in Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 370–377.

[288] Camunas-Mesa, L. A. et al. (2014). "Event-driven stereo vision with orientation filters". In: *Int. Symp. Circuits Syst. (ISCAS)*, pp. 257–260.

[289] Valeiras, David Reverter et al. (2016). "Neuromorphic Event-Based 3D Pose Estimation". In: *Frontiers in Neuroscience* 9.

[290] Xie, Zhen, Shengyong Chen, and Garrick Orchard (2017). "Event-based stereo depth estimation using belief propagation". In: *Frontiers in Neuroscience* 11, p. 535.

[291] Xie, Zhen, Jianhua Zhang, and Pengfei Wang (2018). "Event-based stereo matching using semiglobal matching". In: *International Journal of Advanced Robotic Systems* 15.1.

[292] Felzenszwalb, Pedro F. and Daniel P. Huttenlocher (2004). "Efficient belief propagation for early vision". In: *Conf. on Computer Vision and Pattern Recognition (CVPR)* 1.

[293] Kogler, Jürgen et al. (2014). "Enhancement of sparse silicon retina-based stereo matching using belief propagation and two-stage postfiltering". In: *Journal of Electronic Imaging* 23.4, p. 043011.

[294] Andreopoulos, Alexander et al. (2018). "A Low Power, High Throughput, Fully Event-Based Stereo System". In: *Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 7532–7542.

[295] Zhu, Alex Zihao, Yibo Chen, and Kostas Daniilidis (2018). "Realtime Time Synchronized Event-based Stereo". In: *ECCV*.

[296] Eibensteiner, Florian, Jürgen Kogler, and Josef Scharinger (2014). "A High-Performance Hardware Architecture for a Frameless Stereo Vision Algorithm Implemented on a FPGA Platform". In: *Workshop in CVPR*, pp. 637–644.

[297] Sharma, Kajal, Sung Gaun Kim, and Manu Pratap Singh (2012). "An improved feature matching technique for stereo vision applications with the use of self-organizing map". In: *Int. J. of Precision Engineering and Manufacturing* 13.8, pp. 1359–1368.

[298] Han, Xufeng et al. (2015). "MatchNet: Unifying feature and metric learning for patch-based matching". In: *Conf. on Computer Vision and Pattern Recognition (CVPR)* 07-12-June, pp. 3279–3286.

[299] Hartmann, Wilfried et al. (2017). “Learned Multi-Patch Similarity”. In: *Int. Conf. on Computer Vision (ICCV)*, pp. 1595–1603.

[300] Zbontar, Jurě and Yann Lecun (2016). “Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches”. In: *Journal of Machine Learning Research* 17, pp. 1–32.

[301] Kar, Abhishek, Christian Häne, and Jitendra Malik (2017). “Learning a Multi-View Stereo Machine”. In: *NIPS*.

[302] Weikersdorfer, David, Raoul Hoffmann, and Jörg Conradt (2013). “Simultaneous localization and mapping for event-based vision systems”. In: *Lecture Notes in Computer Science* 7963 LNCS, pp. 133–142.

[303] Kim, Hanme et al. (2014). “Simultaneous mosaicing and tracking with an event camera”. In: *British Machine Vision Conf.*

[304] Milford, Michael et al. (2015). “Towards Visual SLAM with Event-based Cameras”. In: *Robotics Science and Systems conference*.

[305] Zbontar, Jurě and Yann Lecun (2015). “Computing the Stereo Matching Cost with a Convolutional Neural Network”. In: *Conf. on Computer Vision and Pattern Recognition (CVPR)*.

[306] Longuet-Higgins, H.C. (1987). “A computer algorithm for reconstructing a scene from two projections”. In: *Readings in Computer Vision*, pp. 61–62.

[307] Grèzes, J. et al. (2003). “Activations related to "mirror" and "canonical" neurones in the human brain: An fMRI study”. In: *NeuroImage* 18.4, pp. 928–937.

[308] Bonini, Luca et al. (2014). “Space-Dependent Representation of Objects and Other’s Action in Monkey Ventral Premotor Grasping Neurons”. In: *Journal of Neuroscience* 34.11, pp. 4108–4119.

[309] Ott, Benjamin H (2012). “A Convergence Criterion For Self-Organizing Maps”. Master of Science. University of Rhode Island.

[310] Carey, Susan (2004). “Bootstrapping and the origin of concepts”. In: *Daedalus* 133.1, pp. 59–66.

[311] Fischler, Martin A and Robert C Bolles (1981). “Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography”. In: *Graphics and Image Processing* 24.6. Ed. by J.D. Foley, pp. 381–395.

[312] Yuen, HK et al. (1990). “Comparative study of Hough Transform methods for circle finding”. In: *Image and Vision Computing* 8.1, pp. 71–77.

[313] Hyndman, Rob J. and Anne B. Koehler (2006). “Another look at measures of forecast accuracy”. In: *International Journal of Forecasting* 22.4, pp. 679–688.

[314] Song, Rihui et al. (2018). “Calibration of Event-based Camera and 3D LiDAR”. In: *Symp. on Advanced Robotics and Automation*, pp. 102–107.

[315] Dominguez-Morales, Manuel J. et al. (2019). “Bio-Inspired Stereo Vision Calibration for Dynamic Vision Sensors”. In: *IEEE Access* 7, pp. 138415–138425.

[316] Muglikar, Manasi et al. (2021). “How to Calibrate Your Event Camera”. In: *Workshop in Conf. on Computer Vision and Pattern Recognition (CVPR)*.

[317] Huang, Kun, Yifu Wang, and Laurent Kneip (2021). “Dynamic Event Camera Calibration”. In: *Int. Conf. on Intelligent Robots and Systems (IROS)*.

[318] Mueggler, Elias, Basil Huber, and Davide Scaramuzza (2014). “Event-based, 6-DOF pose tracking for high-speed maneuvers”. In: *Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 2761–2768.

[319] Prophesee. *Calibration — Metavision Intelligence Docs 3.0.2*.

[320] Gehrig, Mathias et al. (2021). “DSEC: A Stereo Event Camera Dataset for Driving Scenarios”. In: *IEEE Robotics and Automation Letters* 6.3, pp. 4947–4954.

[321] Yan, Di et al. (2018). “An efficient sparse-dense matrix multiplication on a multicore system”. In: *Int. Conf. on Communication Technology*, pp. 1880–1883.

[322] Latombe, Jean-Claude (1991). *Robot Motion Planning*. Boston, MA: Springer US.

[323] LaValle, Steven (2006). *Planning Algorithms*. Ed. by University of Illinois. Cambridge University Press.

[324] LaValle, S M (2011). “Motion Planning”. In: *IEEE Robotics & Automation Magazine* 18.1, pp. 79–89.

[325] Lynch, Kevin (Kevin M.) and Frank C. Park (2017). *Modern robotics : mechanics, planning, and control*, p. 528.

[326] Choset, Howie M. (2005). *Principles of robot motion : theory, algorithms, and implementation*. MIT Press, p. 603.

[327] Hermann, A. et al. (2013). “Hardware and software architecture of the bimanual mobile manipulation robot HoLLiE and its actuated upper body”. In: *Int. Conf. on Advanced Intelligent Mechatronics*. IEEE, pp. 286–292.

[328] Ju, Zhangfeng, Chenguang Yang, and Hongbin Ma (2014). “Kinematics modeling and experimental verification of baxter robot”. In: *Proceedings of the 33rd Chinese Control Conference*. IEEE, pp. 8518–8523.

[329] Asfour, Tamim et al. (2018). “ARMAR-6: A Collaborative Humanoid Robot for Industrial Environments”. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, pp. 447–454.

[330] Ng, James and Thomas Bräunl (2007). “Performance comparison of Bug navigation algorithms”. In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 50.1, pp. 73–84.

[331] Lozano-Perez, Tomfis and Michael A Wesley (1979). “An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles”. In: *Scientific Applications*.

[332] Ó’Dúnlaing, Colm and Chee K. Yap (1985). “A "retraction" method for planning the motion of a disc”. In: *Journal of Algorithms* 6.1, pp. 104–111.

[333] Takahashi, Osamu and R. J. Schilling (1989). “Motion Planning in a Plane Using Generalized Voronoi Diagrams”. In: *IEEE Transactions on Robotics and Automation* 5.2, pp. 143–150.

[334] Petrovic, Luka (2018). “Motion planning in high-dimensional spaces”. In: *ArXiv* abs/1806.0.

[335] Brock, Oliver and Lydia E. Kavraki (2000). "Towards Real-Time Motion Planning in High Dimensional Spaces". In: *Int. Symp. on Robotics and Automation*.

[336] Karur, Karthik et al. (2021). "A Survey of Path Planning Algorithms for Mobile Robots". In: *Vehicles* 3.3, pp. 448–468.

[337] Costa, Márcia M. and Manuel F. Silva (2019). "A Survey on Path Planning Algorithms for Mobile Robots". In: *Int. Conf. on Autonomous Robot Systems and Competitions (ICARSC)* 19.

[338] Yang, Liang et al. (2016). "Survey of Robot 3D Path Planning Algorithms". In: *J. of Control Science and Engineering*.

[339] Masehian, Ellips and Davoud Sedighizadeh (2007). "Classic and Heuristic Approaches in Robot Motion Planning-A Chronological Review". In: *Int. J. of Mechanical and Mechatronics Engineering* 1.5, pp. 228 –233.

[340] Lengyel, Jed et al. (1990). "Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware". In: *ACM SIGGRAPH Computer Graphics* 24.4, pp. 327–335.

[341] Murray, Don and Cullen Jennings (1997). "Stereo vision based mapping and navigation for mobile robots". In: *Int. Conf. on Robotics and Automation* 2, pp. 1694–1699.

[342] Zidane, Issa Mtanos and Khalil Ibrahim (2018). "Wavefront and a-star algorithms for mobile robot path planning". In: *Advances in Intelligent Systems and Computing* 639, pp. 69–80.

[343] Pal, Anshika, Ritu Tiwari, and Anupam Shukla (2011). "A focused wave front algorithm for mobile robot path planning". In: *Lecture Notes in Computer Science* 6678 LNAI.PART 1, pp. 190–197.

[344] Ghai, Bhavya and Anupam Shukla (2016). "Wave Front Method Based Path Planning Algorithm for Mobile Robots". In: *Int. Conf. on Information and Communication Technology for Intelligent Systems:* 1, pp. 279–286.

[345] Barry, Jennifer L. (2013). "Manipulation with Diverse Actions". PhD thesis.

[346] Pan, Jia and Dinesh Manocha (2015). "Efficient Configuration Space Construction and Optimization for Motion Planning". In: *Engineering* 1, pp. 046–057.

[347] Liu, Yizhou et al. (2021). "Creating Better Collision-Free Trajectory for Robot Motion Planning by Linearly Constrained Quadratic Programming". In: *Frontiers in Neurorobotics* 15, p. 104.

[348] Elbanhawi, Mohamed and Milan Simic (2014). "Sampling-based robot motion planning: A review". In: *IEEE Access* 2, pp. 56–77.

[349] Atas, Fetullah, Lars Grimstad, and Grzegorz Cielniak (2021). "Evaluation of Sampling-Based Optimizing Planners for Outdoor Robot Navigation". In: *Int. Conf. on Intelligent Autonomous Systems (IAS)* 17.

[350] Khatib, O. (1985). "Real-time obstacle avoidance for manipulators and mobile robots". In: *Int. Conf. on Robotics and Automation*, pp. 500–505.

[351] Qureshi, A. et al. (2018). "Motion Planning Networks". In: *Int. Conf. on Robotics and Automation (ICRA)*.

[352] Qureshi, Ahmed Hussain et al. (2021). "Motion Planning Networks: Bridging the Gap Between Learning-Based and Classical Motion Planners". In: *IEEE Transactions on Robotics* 37.1, pp. 48–66.

[353] Chen, Binghong et al. (2020). "Learning to Plan in High Dimensions via Neural Exploration-Exploitation Trees". In: *Int. Conf. on Learning Representations* 11.

[354] Wang, Jiankun et al. (2020). "Neural RRT: Learning-Based Optimal Path Planning". In: *IEEE Transactions on Automation Science and Engineering* 17.4, pp. 1748–1758.

[355] Park, Sun Oh, Min Cheol Lee, and Jaehyung Kim (2020). "Trajectory Planning with Collision Avoidance for Redundant Robots Using Jacobian and Artificial Potential Field-based Real-time Inverse Kinematics". In: *Int. J. of Control, Automation and Systems* 18.8, pp. 2095–2107.

[356] Adar, Nurettin Gökhan (2021). "Real Time Control Application of the Robotic Arm Using Neural Network Based Inverse Kinematics Solution". In: *Sakarya University Journal of Science* 25.3, pp. 849–857.

[357] Zhu, Delong et al. (2018). "Deep Reinforcement Learning Supervised Autonomous Exploration in Office Environments". In: *Int. Conf. on Robotics and Automation (ICRA)*, pp. 7548–7555.

[358] Niroui, Farzad et al. (2019). "Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments". In: *IEEE Robotics and Automation Letters* 4.2, pp. 610–617.

[359] Chen, Fanfei et al. (2019). "Self-learning exploration and mapping for mobile robots via deep reinforcement learning". In: *AIAA Scitech Forum*.

[360] Li, Haoran and Qichao Zhang (2020). "Deep Reinforcement Learning based Automatic Exploration for Navigation in Unknown Environment". In: *IEEE Transactions on Neural Networks and Learning Systems* 31, pp. 2064–2076.

[361] Henderson, Peter et al. (2018). "Deep reinforcement learning that matters". In: *Conf. on Artificial Intelligence (AAAI)*, pp. 3207–3214.

[362] Schmid, Lukas et al. (2022). "Fast and Compute-Efficient Sampling-Based Local Exploration Planning via Distribution Learning". In: *IEEE Robotics and Automation Letters* 7.3.

[363] Christensen, Dennis V et al. (2022). "2022 roadmap on neuromorphic computing and engineering". In: *Neuromorphic Computing and Engineering* 2.2, p. 022501.

[364] Sharifshazileh, Mohammadali et al. (2021). "An electronic neuromorphic system for real-time detection of high frequency oscillations (HFO) in intracranial EEG". In: *Nature Communications* 12.1, pp. 1–14.

[365] Tang, Guangzhi, Arpit Shah, and Konstantinos P. Michmizos (2019). "Spiking Neural Network on Neuromorphic Hardware for Energy-Efficient Unidimensional SLAM". In: *Int. Conf. on Intelligent Robots and Systems*, pp. 4176–4181.

[366] Blouw, Peter et al. (2018). "Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware". In: *Annual Neuro-inspired Computational Elements Workshop* 7, pp. 1–8.

[367] Koziol, Scott, Stephen Brink, and Jennifer Hasler (2014). "A neuromorphic approach to path planning using a reconfigurable neuron array IC". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.12, pp. 2724–2737.

[368] Rueckert, Elmar et al. (2016). "Recurrent Spiking Networks Solve Planning Tasks". In: *Scientific Reports* 6.February, p. 21142.

[369] Ficuciello, Fanny et al. (2014). "Postural synergies of the UB Hand IV for human-like grasping". In: *Robotics and Autonomous Systems* 62.4, pp. 515–527.

[370] D'Avella, Andrea, Philippe Saltiel, and Emilio Bizzi (2003). "Combinations of muscle synergies in the construction of a natural motor behavior". In: *Nature Neuroscience* 6.3, pp. 300–308.

[371] Chen, Pengfei et al. (2018). "Dimensionality Reduction for Motion Planning of Dual-arm Robots". In: *Int. Conf. on Mechatronics and Automation (ICMA)*, pp. 718–723.

[372] Schaal, Stefan et al. (2005). "Learning movement primitives". In: *Springer Tracts in Advanced Robotics* 15, pp. 561–572.

[373] Ijspeert, Auke Jan et al. (2013). "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors". In: *Neural Computation* 25.2, pp. 328–373.

[374] Sayers, Craig (1991). *Self Organizing Feature Maps and Their Applications to Robotics*. Tech. rep. The University of Pennsylvania.

[375] Saxon, James Bennett and Amitabha Mukerjee (1990). "Learning the motion map of a robot arm with neural networks". In: *IJCNN*, pp. 777–782.

[376] Barreto, Guilherme de A., Aluizio F. R. Araújo, and Helge J. Ritter (2003). "Self-Organizing Feature Maps for Modeling and Control of Robotic Manipulators". In: *J. Intell. Robot. Syst* 36.4, pp. 407–450.

[377] Martinetz, Thomas M. and Klaus J. Schulten (1990). "Hierarchical neural net for learning control of a robot's arm and gripper". In: *Int. Joint Conf. on Neural Networks*, pp. 747–752.

[378] Martinetz, Thomas, Helge J. Ritter, and Klaus J. Schulten (1990). "Learning of Visuomotor Coordination of a Robot Arm with Redundant Degrees of Freedom". In: *Parallel Processing in Neural Systems and Computers*.

[379] Behera, L., M. Gopal, and S. Chaudhury (1995). "Self-organizing neural networks for learning inverse dynamics of robot manipulator". In: *Int. Conf. on Industrial Automation and Control*, pp. 457–460.

[380] Benante, Ruben C. and Aluizio F.R. Araújo (2007). "Self-organizing maps to generate state trajectories of manipulators". In: *Int. Conf. on Systems, Man and Cybernetics*, pp. 1590–1595.

[381] Kumar, Swagat et al. (2010). "Visual motor control of a 7DOF redundant manipulator using redundancy preserving learning network". In: *Robotica* 28.6, pp. 795–810.

[382] Ritter, Helge, Thomas Martinetz, and Klaus Schulten (1992). "Visuomotor Coordination of a Robot Arm". In: *Neural Computation and Self-Organizing Maps - An Introduction*. New York. Chap. 11, pp. 156–187.

[383] Ritter, Helge J., Thomas M. Martinetz, and Klaus J. Schulten (1989). "Topology conserving maps for learning visuo-motor-coordination". In: *Neural Networks* 2.3, pp. 159–168.
[384] Wise, K. D. and A. Bowyer (2000). "A survey of global configuration-space mapping techniques for a single robot in a static environment". In: *Int. J. of Robotics Research* 19.8, pp. 762–779.
[385] Brost, R.C. (1989). "Computing metric and topological properties of configuration space obstacles." In: *Int. Conf. on Robotics and Automation (ICRA)*, pp. 170–176.
[386] Zhao, C. S., M. Farooq, and M. M. Bayoumi (1995). "Analytical solution for configuration space obstacle computation and representation". In: *Industrial Electronics Conference (IECON)* 2, pp. 1278–1283.
[387] Varadhan, Gokul et al. (2006). "Topology preserving approximation of free configuration space". In: *Int. Conf. on Robotics and Automation*, pp. 3041–3048.
[388] Ward, James and Jayantha Katupitiya (2007). "Free space mapping and motion planning in configuration space for mobile manipulators". In: *Int. Conf. on Robotics and Automation (ICRA)*, pp. 4981–4986.
[389] Han, Baoling et al. (2021). "Research on Obstacle Avoidance Motion Planning Technology of 6-DOF Manipulator". In: *Adv. Intell. Syst. Comput.* 1296, pp. 604–614.
[390] Wu, Xiaojun, Qing Lit, and K. H. Heng (2005). "A new algorithm for construction of discretized configuration space obstacle and collision detection of manipulators". In: *ICAR*, pp. 90–95.
[391] Xie, Yangmin, Rui Zhou, and Yusheng Yang (2020). "Improved distorted configuration space path planning and its application to robot manipulators". In: *Sensors* 20.21, pp. 1–23.
[392] Huerta-Chua, Jesus et al. (2021). "Exploring a Novel Multiple-Query Resistive Grid-Based Planning Method Applied to High-DOF Robotic Manipulators". In: *Sensors* 21.9.
[393] Davies, Mike (2019a). "Advancing neuromorphic computing From promise to Competitive technology Neuro-Inspired Computational Elements Workshop". In: *Neuro-Inspired Computational Elements (NICE) Workshop*.
[394] Cordel, Macario O. and Arnulfo P. Azcarraga (2015). "Fast Emulation of Self-organizing Maps for Large Datasets". In: *Procedia Computer Science* 52.1, pp. 381–388.
[395] Terlemez, Ömer et al. (2015). "Master Motor Map (MMM) - Framework and toolkit for capturing, representing, and reproducing human motion on humanoid robots". In: *Int. Conf. Humanoid Robots*, pp. 894–901.
[396] Coleman, David et al. (2014). "Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study". In: *arXiv*.
[397] Thomas, Dirk, William Woodall, and Esteve Fernandez (2014). "Next generation ROS: Building on DDS". In: *ROSCon*.
[398] Kam, Hyeong Ryeol et al. (2015). "RViz: a toolkit for real domain data visualization". In: *Telecommunication Systems* 60.2, pp. 337–345.

[399] Şucan, Ioan A., Mark Moll, and Lydia Kavraki (2012). "The open motion planning library". In: *IEEE Robot Autom Mag* 19.4, pp. 72–82.

[400] Kingston, Zachary, Mark Moll, and Lydia E. Kavraki (2019). "Exploring implicit spaces for constrained sampling-based planning". In: *Int. J. of Robotics Research* 38.10-11, pp. 1151–1178.

[401] Strickert, Marc and Barbara Hammer (2003). "Neural Gas for Sequences". In: *WSOM*, pp. 53–57.

[402] Kucuk, Serdar and Zafer Bingul (2006). "Robot Kinematics: Forward and Inverse Kinematics". In: *IndustrialRobotics: Theory, Modelling and Control*. Ed. by Sam Cubero.

[403] Huang, Chung Yang, Chao Yue Lai, and Kwang Ting Cheng (2009). "Fundamentals of Algorithms". In: *Electronic Design Automation*, pp. 173–234.

[404] Ravankar, Abhijeet et al. (2018). "Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges". In: *Sensors* 18.9.

[405] Hermann, Andreas et al. (2014). "Unified GPU voxel collision detection for mobile manipulation planning". In: *Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 4154–4160.

[406] Quigley, Morgan et al. (2009). "ROS: An open-source Robot Operating System". In: *Int. Conf. on Robotics and Automation (ICRA)*.

[407] Bing, Z. et al. (2018). "A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks". In: *Front. in Neurorobotics* 12.

[408] Friedmann, Simon et al. (2016). "Demonstrating Hybrid Learning in a Flexible Neuromorphic Hardware System". In: *Trans. Biomed. Circuits Syst.* 11.1, pp. 128–142.

[409] Vineyard, Craig M. et al. (2019). "Benchmarking event-driven neuromorphic architectures". In: *ACM International Conference Proceeding Series*.

[410] Rowley, A. et al. (2020). "Stacks of Software Stacks". In: *SpiNNaker – A Spiking Neural Network Architecture*. Ed. by Steve Furber and Petruţ Bogdan. Now Publishers Inc., pp. 79–128.

[411] Garside, James and Luis A. Plana (2020). "The SpiNNaker chip". In: *SpiNNaker – a spiking neural network architecture*, pp. 17–52.

[412] Owens, John D. et al. (2008). "GPU computing". In: *Proceedings of the IEEE* 96.5, pp. 879–899.

[413] Molla, Md Mamun et al. (2018). "GPU Accelerated Multiple-Relaxation-Time Lattice Boltzmann Simulation of Convective Flows in a Porous Media". In: *Frontiers in Mechanical Engineering* 4.

[414] Whitehead, Nathan (2011). *Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs*. Tech. rep.

[415] Brette, Romain et al. (2007). "Simulation of networks of spiking neurons: A review of tools and strategies". In: *Journal of Computational Neuroscience* 23.3, pp. 349–398.

[416] Goodman, Dan F.M. and Romain Brette (2009). "The brian simulator". In: *Frontiers in Neuroscience* 3.SEP, pp. 192–197.

[417] Pehle, Christian and Jens Egholm Pedersen. *Norse - A deep learning library for spiking neural networks*.

[418] Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High Performance Deep Learning Library". In: *Int. Conf. on Neural Information Processing Systems (NIPS)* 721, pp. 8026–8037.

[419] Carnevale, N. T. and Hines M. L. (2006). *The NEURON Book | NEURON*. Cambridge University Press.

[420] Gewaltig, Marc-Oliver and Markus Diesmann (2007). "NEST (NEural Simulation Tool)". In: *Scholarpedia* 2.4, p. 1430.

[421] Bower, James M. et al. (1998). "Introduction". In: *The Book of GENESIS*. Springer New York, pp. 3–5.

[422] Stimberg, Marcel, Romain Brette, and Dan F.M. Goodman (2019). "Brian 2, an intuitive and efficient neural simulator". In: *eLife* 8.

[423] Rittner, Pedro and Thomas A Cleland (2014). *Myriad : a transparently parallel GPU-based simulator for densely integrated biophysical models Implementational Details Planned Extensions Why another simulator?* Tech. rep.

[424] Davison, Andrew P et al. (2008). "PyNN: A Common Interface for Neuronal Network Simulators." In: *Frontiers in neuroinformatics* 2, p. 11.

[425] Eppler, Jochen Martin et al. (2009). "PyNEST: A Convenient Interface to the NEST Simulator". In: *Frontiers in neuroinformatics* 2.

[426] Morrison, Abigail et al. (2005). "Advancing the boundaries of high connectivity network simulation with distributed computing". In: *Neural computation* 17.8, pp. 1776–1801.

[427] Rotter, Stefan and Markus Diesmann (1999). "Exact digital simulation of time-invariant linear systems with applications to neuronal modeling". In: *Biological Cybernetics* 81.5-6, pp. 381–402.

[428] MacGregor, Ronald J. (1987). *Neural and brain modeling*. Academic Press, p. 643.

[429] Vitay, Julien, Helge Dinkelbach, and Fred H. Hamker (2015). "ANNarchy: A code generation approach to neural simulations on parallel hardware". In: *Frontiers in Neuroinformatics* 9, p. 19.

[430] Potjans, Tobias C. and Markus Diesmann (2014). "The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model". In: *Cerebral Cortex* 24.3, pp. 785–806.

[431] Schmuker, Michael and Gisbert Schneider (2007). "Processing and classification of chemical data inspired by insect olfaction". In: *Proceedings of the National Academy of Sciences of the United States of America* 104.51, pp. 20285–20289.

[432] Diamond, Alan et al. (2014). "Classifying chemical sensor data using GPU-accelerated bio-mimetic neuronal networks based on the insect olfactory system". In: *BMC Neuroscience* 15.S1, pp. 1–2.

[433] Rhodes, Oliver et al. (2018). "Spynnaker: A software package for running pynn simulations on spinnaker". In: *Frontiers in Neuroscience* 12, p. 816.

[434] Franklin, Dustin (2018). *NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics*.

[435] Zehra, Farzeen et al. (2020). "Comparative Analysis of C++ and Python in Terms of Memory and Time".

[436] *EVALUATION KIT - Gen3 HVGA-EM* (2022).

[437] *RealSense Depth Camera D435* (2022).

Publications by the author et al.

Steffen, Lea et al. (2019a). “Creating an obstacle memory through event-based stereo vision and robotic proprioception”. In: *Int. Conf. on Automation Science and Engineering (CASE)* 15, pp. 1829–1836.

Steffen, Lea et al. (2019c). “Neuromorphic Stereo Vision: A Survey of Bio-Inspired Sensors and Algorithms”. In: *Frontiers in Neurorobotics* 13, p. 28.

Kaiser, Jacques et al. (2018). “Microsaccades for neuromorphic stereo vision”. In: *Lecture Notes in Computer Science* 11139, pp. 244–252.

Steffen, Lea et al. (2021c). “Reducing the Dimension of the Configuration Space with Self Organizing Neural Networks”. In: *Int. Conf. on Advanced Robotics and Mechatronics (ICARM)*.

Steffen, Lea et al. (2019b). “Multi-view 3D reconstruction with self-organizing maps on event-based data”. In: *Int. Conf. on Advanced Robotics, (ICAR)* 19.

Steffen, Lea et al. (2021a). “A Benchmark Environment for Neuromorphic Stereo Vision”. In: *Frontiers in Robotics and AI, section Robot and Machine Vision*.

Steffen, Lea et al. (2022a). “Comparing SONN Types for Efficient Robot Motion Planning in the Configuration Space”. In: *Int. Conf. on Intelligent Autonomous Systems (IAS)* 17.

Steffen, Lea et al. (2022b). “Reactive Neural Path Planning with Dynamic Obstacle Avoidance in a Condensed Configuration Space”. In: *Int. Conf. on Intelligent Robots and Systems (IROS)*.

Steffen, Lea et al. (2020b). “Networks of Place Cells for Representing 3D Environments and Path Planning”. In: *Int. Conf. on Biomedical Robotics and Biomechatronics (BioRob)* 7.

Steffen, Lea et al. (2020a). “Adaptive, Neural Robot Control –Path Planning on 3D Spiking Neural Networks”. In: *Int. Conf. on Artificial Neural Networks (ICANN)* 29.

Steffen, Lea et al. (2021b). “Benchmarking Highly Parallel Hardware for Spiking Neural Networks in Robotics”. In: *Frontiers in Neuroscience* 15, p. 790.

Related student work

Hauck, Bendikt (2019). “Towards an obstacle memory: A voxel-based approach to model a robot’s proprioception using spiking neurons”. Master. Karlsruher Institut für Technologie.

Augenstein, Philipp (2021). “Configuration aware Neural Path Planning in the Task Space”. Master. Hochschule Karlsruhe University of Applied Sciences.

Glueck, Katharina (2021). “Reducing the Dimensionality of the Configuration Space with Self Organizing Neural Networks”. Master. Karlsruher Institut für Technologie.

Ehrlinspiel, Björn (2019). “Ereignisbasierte Tiefenwahrnehmung mit Hilfe einer Self-Organizing Map”. Bachelor. Karlsruher Institut für Technologie.

Elfgen, Max (2020). “A Benchmark Environment for Stereo Vision on Event-Based Data”. Master. Karlsruher Institut für Technologie.

Azanov, Daniel (2022). “A Plausible Computational Model for Emulating Early Stereo Vision Development through Self-Organization and Bootstrapping”. Master. Karlsruher Institut für Technologie.

Weyer, Tobias J. (2021). “Collision-Free Path Planning in the Reduced Configuration Space of Robots by Self-Organizing Neural Networks”. Master. Karlsruher Institut für Technologie.

Liebert, Artur (2019). “Der Wavefrontalgorithmus auf Modifizierten Puls-Gekoppelten Neuronalen Netzen”. Master. Karlsruher Institut für Technologie.

Koch, Robin (2020). “Benchmarking High-Performance Parallel Architectures using Robotic Applications”. Bachelor. Karlsruher Institut für Technologie.