A Neuromorphic Approach to Obstacle Avoidance in Robot Manipulation

Ahmed Faisal Abdelrahman

Publisher: Dean Prof. Dr. Sascha Alda

Hochschule Bonn-Rhein-Sieg – University of Applied Sciences,
Department of Computer Science

Sankt Augustin, Germany

November 2023

Technical Report 02-2023

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSN 1869-5272

ISBN 978-3-96043-111-4

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

This work was supervised by	Paul G. Plöger Maren Bennewitz Matias Valdenegro-Toro

We gratefully acknowledge the support
by the b-it International Center for Information Technology.

Digital Object Identifier https://doi.org/10.18418/978-3-96043-111-4

This technical report is derived from the author's Master's thesis report. The few significant modifications to the original text are highlighted in blue.

Abstract

Neuromorphic computing aims to mimic the computational principles of the brain *in silico* and has motivated research into event-based vision and spiking neural networks (SNNs). Event cameras (ECs) capture local, independent changes in brightness, and offer superior power consumption, response latencies, and dynamic ranges compared to frame-based cameras. SNNs replicate neuronal dynamics observed in biological neurons and propagate information in sparse sequences of "spikes". Apart from biological fidelity, SNNs have demonstrated potential as an alternative to conventional artificial neural networks (ANNs), such as in reducing energy expenditure and inference time in visual classification. Although potentially beneficial for robotics, the novel event-driven and spike-based paradigms remain scarcely explored outside the domain of aerial robots.

To investigate the utility of brain-inspired sensing and data processing in a robotics application, we developed a neuromorphic approach to real-time, online obstacle avoidance on a manipulator with an onboard camera. Our approach adapts high-level trajectory plans with reactive maneuvers by processing emulated event data in a convolutional SNN, decoding neural activations into avoidance motions, and adjusting plans in a dynamic motion primitive formulation. We conducted simulated and real experiments with a Kinova Gen3 arm performing simple reaching tasks involving static and dynamic obstacles. Our implementation was systematically tuned, validated, and tested in sets of distinct task scenarios, and compared to a non-adaptive baseline through formalized quantitative metrics and qualitative criteria.

The neuromorphic implementation facilitated reliable avoidance of imminent collisions in most scenarios, with 84% and 92% median success rates in simulated and real experiments, where the baseline consistently failed. Adapted trajectories were qualitatively similar to baseline trajectories, indicating low impacts on safety, predictability and smoothness criteria. Among notable properties of the SNN were the correlation of processing time with the magnitude of perceived motions (captured in events) and robustness to different event emulation methods. Preliminary tests with a DAVIS346 EC showed similar performance, validating our experimental event emulation method. These results motivate future efforts to incorporate SNN learning, utilize neuromorphic processors, and target other robot tasks to further explore this approach.

Acknowledgements

I am indebted to my supervisors for their guidance throughout this thesis project. Firstly, I would like to express my appreciation to Prof. Plöger, from whom I have learned much over the years, for stimulating discussions, an encouraging attitude, and igniting my interest in various problems and methods. I am grateful to Prof. Bennewitz for expressing interest in the concepts I developed in this thesis and for undertaking my supervision. I would like to especially thank Prof. Valdenegro for his advice, suggestions, and involvement long before the beginning of this project and to the very end. This work may have never materialized if not for his initial support and interest in my research ideas.

My sincerest gratitude and thanks go to my family, especially my parents, to whom I owe everything, and whose endless encouragement and support have been a primary source of motivation.

Contents

List of Symbols

Symbol	Description
t	Time
$L(\mathbf{x}_k, t_k)$	Intensity at pixel $\mathbf{x}_k$, time t_k
e_k	Event k
p_k	Polarity of event k
$\mathbf{y}$	Position in 3D space
$\mathbf{Y}$	Trajectory in 3D space
ζ	Estimated angular velocity
θ	Event emission threshold
s_{BE}	Binary erosion filter structure size
v	SNN membrane potential
S	Spike train
v_{thresh}	SNN potential spiking threshold
v_{rest}	SNN resting potential
v_{reset}	SNN reset potential
T_{refrac}	SNN refractory spiking period
τ_v	SNN potential decay constant
W	SNN weight matrix
w_c	SNN weight initialization constant
T_{sim}	SNN simulation time period
K	Convolutional SNN Kernel
$\mathbf{g}$	DMP goal position
w	DMP basis function weights
τ	DMP time-scale parameter
s	DMP phase variable
f	DMP forcing term
ψ	DMP basis functions
η	Constant gain (PF; Park)
p_0	Obstacle radius of influence (PF; Park)
C_δ	Gradient constant factor (PF; Park)

Symbol	Description
$\tilde{\boldsymbol{\phi}}$	Mean negative potential gradient (PF)
$\boldsymbol{\phi}$	Obstacle avoidance acceleration term
ϕ_{max}	Upper limit on $\boldsymbol{\phi}$
n_ϕ	$\boldsymbol{\phi}$ history horizon
t_{act}	FST activation threshold factor
f_v	Motion loop frequency
K_i	Controller gain
e	Positional error
$\mathbf{v}(t)$	Applied velocity
$\delta_{safety,1}$	Distance tolerance (1)
$\gamma_{v,1}$	Velocity reduction factor (1)
$\gamma_{a,1}$	Acceleration reduction factor (1)
$\delta_{safety,2}$	Distance tolerance (2)
$\gamma_{v,2}$	Velocity reduction factor (2)
$\gamma_{a,2}$	Acceleration reduction factor (2)
$\delta^+_{pos,i}$	Upper positional limit (dimension i)
$\delta^-_{pos,i}$	Lower positional limit (dimension i)
$\delta_{\mathbf{y}}$	Position reaching tolerance
$\delta_{\mathbf{g}}$	Goal reaching tolerance
δ_{obs}	Obstacle avoidance distance tolerance
N_s	Number of scenarios per task
N_{pos}	Number of poses in a trajectory
T	Task execution time
$l_{\mathbf{Y}}$	Trajectory Length
$N_{collisions}$	Number of collisions
d_G	Distance to goal
S	Success
T_{comp}	Computation time

List of Abbreviations

Acronym	Full Form
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
C-SNN	Convolutional Spiking Neural Network
DMP	Dynamic Motion Primitive
DNN	Deep Neural Network
DQN	Deep Q-Learning
DVS	Dynamic Vision Sensor
EC	Event Camera
FOV	Field of View
FST	First-Spike-Time
H-H	Hodgkin and Huxley (spiking model)
HRC	Human-Robot Collaboration
LIF	Leaky Integrate-and-Fire
LSTM	Long Short-Term Memory
OP	Optical Flow
PF	Potential Field
PSP	Post-synaptic Potential
RL	Reinforcement Learning
RNN	Recurrent Neural Network
R-STDP	Reward-modulated spike-timing-dependent plasticity
SNN	Spiking Neural Network
SRM	Spike Response Model
STDP	Spike-timing-dependent plasticity

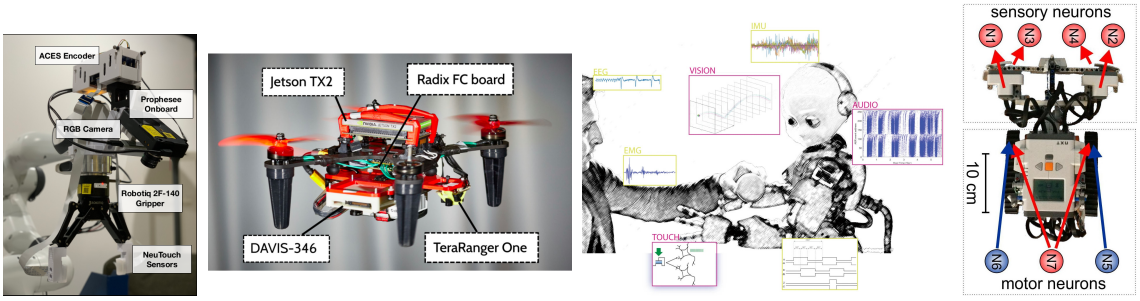
1

Introduction

Modern autonomous systems can excel at targeted tasks, but generally lack capabilities that the average human exemplifies, including rapidly learning for and accomplishing a range of unstructured tasks. One potential reason is the fundamental differences in human and artificial intelligence (AI) due to their respective physical substrates (biological vs. silicon-based) (Korteling et al. (2021)). AI systems likely have access to superior information propagation speeds, communication bandwidths, and raw computational power. Though this may imply a greater capacity for multi-sensory processing, robotic agents embodying AI would still struggle with everyday tasks that our brains facilitate with relative ease. This poses the question of whether the information propagation mechanisms and computational architectures present in our brains could be a key factor in the advancement of reliable autonomy.

The human brain is capable of maintaining multiple, complex cognitive processes and still being more energy-efficient than contemporary computers. It simultaneously regulates essential processes like breathing and heart function, as well as controlling visual perception, language processing and speech, thought, fine and coarse motor activity, etc., all while consuming ∼20W of power (similar to some lightbulbs, Drubach (2000)). Comparable and less-capable computers require power inputs many orders of magnitudes higher (see Appendix A for a comparison of examples). This is evident when simulating cortical neural networks on conventional computers; at the scale of a mouse, such a simulation has been shown to run at 40,000 more power and 9,000 times less speed, while projections from the Human Brain Project underscore the colossal power requirement for simulating a human brain (Thakur et al. (2018)). These and other neuroscientific studies indicate the vastly superior performance-to-efficiency ratio of the brain, and motivates the consideration of biologically-inspired circuitry, sensors, and algorithms in intelligent robot design.

Biological inspiration has frequently driven practical innovations, leading to IR detectors and gyroscopes (Wicaksono (2008)), learning paradigms such as evolutionary algorithms and reinforcement learning (RL) (Sutton et al. (1998)), swarm intelligence algorithms, and locomotion control using elementary neural circuits (Ijspeert (2008)), to name a few. Artificial neural networks (ANNs) are based on their biological counterparts: the earliest models by McCulloch and Pitts consisted of interconnected layers of simple computational units with adjustable "synaptic" connections, while later convolutional neural networks (CNNs) aimed to mimic connectivity patterns observed in animal visual cortices. CNNs have been largely successful in specific visual processing tasks (see Goodfellow et al. (2016) for a comprehensive



(a) Taunyazov et al. (2020)

(b) Sun et al. (2021)

(c) Bartolozzi et al. (2022)

(d) Lobov et al. (2020)

Figure 1.1: Recent examples of neuromorphic robot design: (1.1a) an EC and neuromorphic tactile sensor for manipulation; (1.1b) an EC onboard a quadrotor; (1.1c) the iCub robot equipped with neuromorphic visual, auditory, and tactile sensing; (1.1d) SNNs for learning simple navigation behaviours in a neurologically-inspired sensory-motor architecture.

review), and are commonly deployed on robots. Nevertheless, these models are crude approximations at best. Biological neurons asynchronously aggregate inputs over time and propagate discrete, sparse spikes (action potentials), whose precise timings are thought to encode useful information. Conversely, artificial neurons synchronously and continuously propagate real-valued signals, abandoning an additional temporal dimension afforded by relative spike timings. These discrepancies become relevant following observations that deep neural networks (DNNs), despite their notable success in visual recognition, still suffer from ever-growing numbers of parameters (requiring more computations), correspondingly sizable data requirements, poor generalization to unobserved yet similar inputs, and catastrophic failures in response to minor perturbations (Serre (2019)). Although unbiased comparisons of human and artificial intelligence are not straightforward (Cowley et al. (2022), Firestone (2020)), our brain seems better-equipped to handle an extraordinary range of problems while exhibiting superior generalization and robustness to noise and variance than present AI models.

Apart from neuronal design, DNNs depart from biology in their primary approach to learning: the popular back-propagation algorithm. The gradient-based synaptic adjustment rule has been instrumental in the aforementioned success, but is largely considered biologically implausible. Next to substantial data and computational overheads, this provides further impetus to look to the brain for inspiration. A general reluctance towards this view may be a product of the meteoric success of CNNs and a long-standing hesitancy to deal with the complexities of the brain (Crick (1989)). Nevertheless, recent studies have demonstrated progress towards more biologically-plausible learning that could lead to similar results (Illing et al. (2019)).

Aside from learning, the field of neuromorphic computing explores more biologically faithful sensors and computational architectures.

Neuromorphic Computing

Neuromorphic engineering/computing aims to reproduce known characteristics of the brain in hardware, particularly in analog VLSI circuits. The field was established in the 1980s by Carver Mead, who suggested analogies between biological neuronal dynamics and the physics of sub-threshold regions of transistor

operation. This has given rise to various models of *spiking neural networks*, dedicated *neuromorphic processors*[1] designed to run them, such as the Intel Loihi, SpiNNaker, and IBM TrueNorth (Bouvier et al. (2019)), and neuromorphic sensors that include *event cameras*. This research is motivated by the pursuit of brain-like computation to improve efficiency, parallelization, and energy consumption (Thakur et al. (2018)) through a fundamental paradigm shift. Neuromorphic, energy-efficient computation could benefit various applications including autonomous vehicles, wearable devices, and IoT[2] sensors (Rajendran et al. (2019)). Naturally, this holds promise for other embodied AI agents and robotic systems as well.

Various studies provide empirical evidence of the advantages of neuromorphic computing: 10 and 1000 factor increases in speed and energy-efficiency, respectively, when learning on a neuro-processor compared to a conventional processor (Wunderlich et al. (2019)), a four-fold increase in the energy-efficiency of a spiking network instead of a DNN for speech recognition (Blouw & Eliasmith (2020)), better energy-per-classification ratios for deep learning problems when comparing to a Tesla P100 GPU (Göltz et al. (2021)), and speed and efficiency gains in various classical problems (Davies et al. (2021)). These results have coincided with, or perhaps fostered, an ongoing interest in the field, typified by the recent establishment of the "Neuromorphic Computing and Engineering" journal (Indiveri (2021)).

Event Cameras

Event cameras (ECs) are a typical example of a neuromorphic sensor and are modelled after biological retinas. ECs exclusively record per-pixel *events* at which the change in pixel intensity crosses a set threshold, mimicking retinal photoreceptor cells (Posch et al. (2014)). Consequently, they only capture significant intensity changes, often due to motion (Figure 1.2), in contrast to frame-based cameras, whose pixels synchronously and continuously transmit absolute intensity values, much of which is often redundant information. Therefore, similar to our retinas, ECs do not capture full images through a periodic shutter-like mechanism as conventional camera do. The idea of only capturing signals of interest is evidently desirable in various scenarios, such as processing speech amongst background noise in recognition applications, or focussing on regions where changes occur in image processing (Blouw & Eliasmith (2020)).

Apart from higher biological fidelity, ECs offer lower power consumption, lower transmission latencies, higher dynamic ranges, and more robustness to motion blur than traditional cameras (Gallego et al. (2022), G. Chen et al. (2020)), which have been experimentally validated (Sun et al. (2021)). Equivalently, these properties can address limitations of frame-based vision in power consumption and bandwidth due to transmitting larger amounts of data (Dubeau et al. (2020)). External clock-driven data acquisition naturally leads to redundant image information and the potential loss of inter-frame information (Risi et al. (2020)). Instead, ECs selectively acquire data based on scene dynamics. ECs have most often been deployed in applications requiring rapid reaction speeds, such as drone flight, due to characteristically high temporal resolutions, but could be beneficial in robot navigation and manipulation.

The asynchronous nature of event-based data necessitates novel methods and algorithms; the artificial analogs of visual cortical cells, spiking neurons, are prime candidates.

[1] Also known as "neuromorphic chips", "neuromorphic emulators", "neural emulators", and "neural processors"
[2] Internet of Things

(a) Fast object (RGB) (b) Fast object (events) (c) Motion 1 (RGB + events) (d) Motion 2 (RGB+events)

Figure 1.2: Images captured from a DAVIS346 EC. 1.2a shows an RGB image of a fast-moving object and 1.2b visualizes the captured events (note the relatively clear effects of motion blur in the first). 1.2c and 1.2d show events superimposed on images captured while moving the camera, where events are most visible at object borders.

Spiking Neural Networks

Spiking neural networks (SNNs) are a neuromorphic alternative to conventional ANNs in which computational units propagate sparse sequences of *spikes*, instead of real-valued signals (Figure 1.3). Input spikes to a neuron contribute to a decaying internal aggregate of past inputs: its *membrane potential*. When a threshold is exceeded, the neuron emits a spike (or *action potential*) which resets its potential. Spiking neurons operate asynchronously, such that information flows through the network through trains of distinctly-timed spikes. These neuronal dynamics match those observed in biology (particularly in the primary visual cortex; G. Chen et al. (2020)), but raise challenges concerning the encoding, decoding, and processing of spiking data as well as learning methods, which remain open areas of research.

Similar to the event/frame-based camera dichotomy, SNNs have stimulated interest due to their

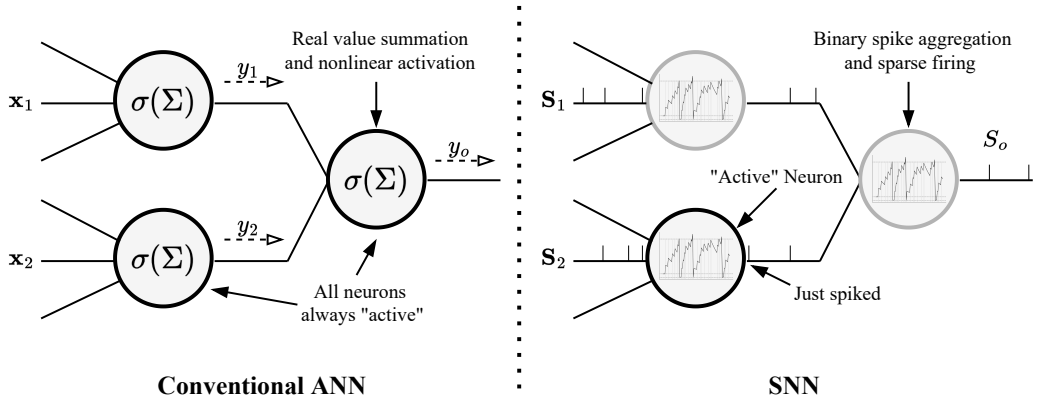


Figure 1.3: An illustration of neuronal dynamics in conventional ANNs (left) and SNNs (right). Neurons in ANNs constantly compute and propagate real-valued signals (input summation and nonlinear activation, such as a sigmoid function), while spiking neurons aggregate spikes and only fire at superthreshold activation. Gray/black borders around neurons signify inactivity/activity; note that only the bottom neuron in the SNN is "active" at this timepoint, since it had just spiked.

potential advantages over ANNs. SNNs can be as, or potentially more, expressive than ANNs (Maass (1997)), and successful applications in common visual tasks have shown that they can consume less power and exhibit faster classification inferences (Neil et al. (2016)), as well as outperform ANNs in energy-delay-product[3] (Davies et al. (2021)). This energy-efficiency is due to neurons emitting outputs only when significantly stimulated, as opposed to constant computations as in ANNs, though this advantage is fully realized with dedicated neuromorphic hardware. This is also reflected in response latencies; in classification problems, SNNs could produce a correct inference before a frame-based approach could fully process all input pixels (Neil et al. (2016)).

Although often associated with embodied agents, SNNs have recently been applied in other areas of research, such as for the contemporary problem of detecting Covid-19 from CT scans (Garain et al. (2021)).

A Neuromorphic Approach to A Robotics Problem

In the pursuit of improving the intelligence and efficiency of autonomous agents, we seek to investigate the application of biologically-inspired sensors and algorithms. This thesis project explores the utility of incorporating neuromorphic elements: event cameras and spiking neural networks, in an approach to a common robotics problem.

The problem we address is obstacle avoidance, which we define here as planning or reacting to prevent collisions with obstacles interfering with a task. Avoiding obstacles is a rudimentary but essential action for robotic as well biological agents, which organisms with relatively simple neuronal architectures such as insects seem to perform exceptionally well (particularly in comparison to robots). Therefore, it provides an adequate context for testing a biologically-inspired approach on a robot platform. In this context, the two neuromorphic elements we focus on have special relevance: event-based vision is particularly appropriate for motion-based perception and action, while SNNs are well-suited to processing the asynchronous, binary signals produced by ECs (since events are essentially equivalent to spikes). We specifically focus on obstacle avoidance in manipulation, for which fewer research has been conducted in comparison to the domains of aerial and mobile robots.

Motivated by the potential merits of a neuromorphic strategy, we aim to demonstrate the viability and utility of a neuromorphic approach through its implementation and by conducting a systematic analysis of its results. Although we are interested in the prospects of transitioning to biologically-inspired components, the intention is not to strictly argue that blind mimicry of biological systems is a definitive, global solution[4], but rather to work towards a better understanding of its potential, benefits and limitations in a robotics application through methodical experimentation.

In the upcoming sections, we formally state the addressed problem, outline the delimitations, and present the contributions of this thesis.

[3] A measure of energy efficiency that incorporates inference delays.

[4] For example, ornithopters are known to be more biologically plausible than modern aircraft, but are demonstrably worse at flight.

1.1 Problem Statement

The aim of this thesis is to conduct research into neuromorphic methods and develop a neuromorphic solution that is applicable to a robotics problem. To that end, event cameras (ECs) and spiking neural networks (SNNs) are central aspects, providing the sensor and neural circuitry, respectively, that constitute our neuromorphic solution. Specifically, we investigate the processing of event data using SNNs to facilitate a visuomotor skill on a robotic manipulator.

The application we target is **real-time**, **online** obstacle avoidance on a camera-equipped manipulator. Although more commonly studied in mobile and aerial robot navigation, obstacle avoidance in manipulation scenarios can be equally useful for increasing safety and task efficiency, especially in unconstrained human-robot collaboration (HRC) scenarios. Ideally, adjustments of planned trajectories must be performed in real-time (quick enough to avoid collisions) and online (during trajectory execution, not prior planning); two properties we particularly consider. As previously expressed, ECs excel at capturing rapid motions, while SNNs are best-suited by design to process event data, thus motivating their utilization for deriving corrective avoidance motions.

Following a thorough experimental evaluation, we analyze the results to validate the hypothetical merits of the approach. These include the efficacy of event data for capturing obstacle motions, and the tendency of SNNs to propagate "relevant" information and consequently reduce unnecessary data processing. As a primary objective, we seek to determine the feasibility of the neuromorphic approach and draw initial, well-grounded insights on its utility in a robotics scenario and thus assess the value of pursuing these concepts further.

The goal is thus a contribution to answering the question: *are more biologically-inspired sensors and algorithms worth studying and applying further in the context of robotics?* In the author's view, the potential value in neuromorphic implementations that we have outlined (and discuss further in the literature review of section 2) warrants dedicating more research into this approach. Through our study, we make progress towards this goal by attempting to address the following relevant questions:

- Are there benefits to using event camera data for obstacle avoidance on a robot?
- Are there benefits to using SNN processing for obstacle avoidance on a robot?
- Is it feasible to decode SNN outputs into obstacle avoidance behaviour on a robot?
- Is it possible to achieve reliable, online obstacle avoidance with the proposed neuromorphic approach (of utilizing ECs in conjunction with SNNs)?

The first challenge in addressing these questions is the design and development of an SNN-based obstacle avoidance method. This method should handle the transformation of visual inputs (event data), into meaningful obstacle avoidance behaviour. We initially focus on processing emulated event data, and design an emulation component that can be easily substituted with a real event camera. Emulating events is convenient for obtaining initial results, particularly in the absence of an EC[5]; while we expect experiment conclusions to carry over, it is important to verify this in subsequent analyses. Subsequently,

[5] An EC was not available in the initial stages of this project, initially making emulation a necessity.

we attempt to integrate a real event camera in our pipeline and conduct preliminary tests for validation. Here, we run SNN simulations on conventional hardware, and leave the integration of neuromorphic processors for future work.

The next challenge involves demonstrating the function of and testing this method on a robot platform. To that end, we test on the Kinova Gen3 robotic arm in simple task scenarios involving static and dynamic obstacles to demonstrate obstacle avoidance behaviour at the end-effector. The end-effector is pre-equipped with a color camera, from which we emulate event data. In later tests, an event camera must be mounted in its stead. We perform initial testing in Gazebo simulations before transferring to the real robot. Starting in simulation enables rapid testing and evaluation during the initial stages of development. As a consequence, we also evaluate how well our method transfers from simulation to the real world.

The third primary challenge is to conduct a thorough evaluation that is conducive to establishing well-grounded conclusions to this study. Due to the novelty of our approach, we have no access to appropriate benchmarks to systematically compare against. As will be shown in section 2, few similar works exist and close examples that involve manipulator obstacle avoidance often lack results in metrics that could quantify performance and provide opportunities for comparisons. This creates a necessity for formulating appropriate quantitative and qualitative metrics and criteria on the basis of which we can adequately assess the results of our proposal.

1.2 Delimitations

Certain elements of our approach and its implementation have been limited or excluded from the scope of this thesis project due to practical considerations that include time constraints. We outline them in this section.

We do not utilize neuromorphic hardware for running SNNs in this study, and instead simulate the networks on conventional (digital) processors. A primary factor is the relative difficulty in acquiring a dedicated neuro-processor at the time of writing, since they are not yet widely and commercially available. In theory, the true potential of SNNs is limited by conventional, synchronous processors; a clock-driven architecture can not ideally simulate the independent analog dynamics within and propagation of spikes from each neuron. Nevertheless, these can be simulated to a level of precision that we expect are satisfactory for the purposes of this study, including drawing conclusions on the utility of the architectural properties of SNNs. A related factor concerns scope limitation, since neuro-processors are expected to introduce hardware-related challenges that may prove time-consuming to address. For the initial demonstration of our neuromorphic approach that we seek to present here, the ability to process spiking data into useful behaviour is ultimately a sufficient outcome. Though we can not adequately investigate them here, the power consumption and other hardware-specific advantages of SNNs can be addressed in future extensions involving neuro-processors.

In a similar vein, we primarily rely on emulating event data from conventional camera data for the main results of this thesis. This provides a convenient substitute for real event camera data, which facilitates the intended evaluation of our approach, though it discards the strict asynchrony of events (see section 4.1.2). Preliminary tests with a real event camera are conducted near the end of the study, but a

more thorough evaluation is intended for future work, and the main conclusions here are drawn in the context of emulated event data.

We do not consider SNN synaptic weight adjustments, i.e. *learning*, in this work. Although this could lead to improvements in the end-to-end performance of our method, we constrain our attention to evaluating the feasibility and success of our approach in order to draw principled conclusions on its utility before premature optimization efforts.

In our evaluations of obstacle avoidance performance, we only consider obstacles within the end-effector camera's field of view (FOV). By its nature, the on-board sensor is incapable of visually perceiving obstacles outside its FOV, just as we could not. Since we focus on evaluating how well perceived events can be processed to produce reactive avoidance motions, we disregard obstacles the robot can not presently perceive. We deliberately avoid the common approach of placing multiple sensors around the robot's workplace. Though this may partially address the issue, it adds an undesirable constraint on a robot's operational space by necessitating the installation of additional sensors in any environment it inhabits; instead, we consider a self-contained solution. It is possible to utilize some form of active vision or maintaining memories of obstacles observed in the recent past, but these functionalities are not necessary for the focus of this thesis and could be added in future, more complete implementations.

We also exclusively consider collisions with the end-effector. Avoiding whole-body collisions may require additional functionalities for remembering and extrapolating relative obstacle positions[6], for example, and incorporating these in future trajectory adjustments. We avoid the complexities of these augmentations, since we can sufficiently evaluate our neuromorphic approach by only avoiding end-effector collisions (and, as mentioned, could improve the implementation in future extensions).

1.3 Contributions

We end this introduction by discussing the main contributions of this work, which can be summarized as follows:

- Providing a review of the literature on the relevant fields of research.
- Designing and implementing an SNN-based method for obstacle avoidance in robot manipulation.
- Formulating quantitative metrics and qualitative criteria for evaluating obstacle avoidance performance and trajectory characteristics.
- Evaluating our approach in simulation and on the real robot through a range of task scenarios.
- Conducting analyses of different event emulation strategies, performance without the SNN component, and the effects of varying SNN weights.
- Conducting preliminary evaluations with a real event camera.

The conceptualization and implementation of our approach is a product of the research conducted on the central topics of this thesis. The notable findings of this research are presented in a review of

[6]Most collisions occurring on the rest of the robot body are likely to happen once the obstacle is outside the camera's FOV.

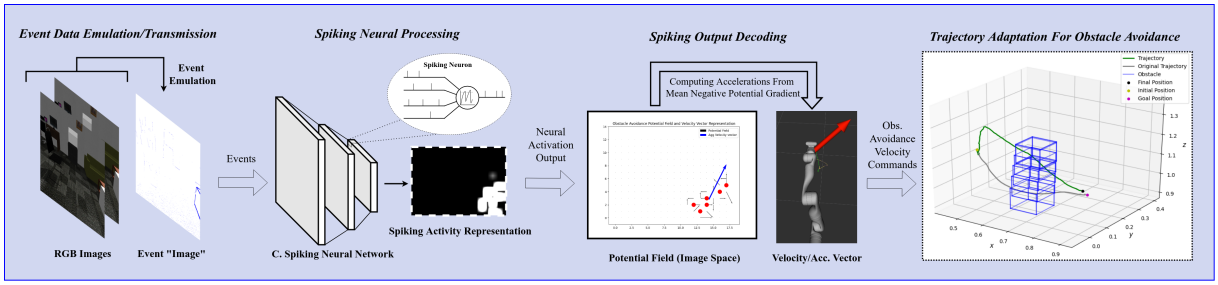


Figure 1.4: An illustration of the main stages of our proposed SNN-based obstacle avoidance pipeline. The plot on the right shows a pre-planned trajectory (grey) and a resultant trajectory (green) adapted to avoid an obstacle (blue).

the literature on event cameras, spiking neural networks, neuromorphic computing, and relevant work on obstacle avoidance. In addition, we provide additional background information on these topics in a dedicated chapter.

Our primary contribution is the design and implementation of a neuromorphic approach to obstacle avoidance on a robot manipulator equipped with an on-board camera. This approach consists of a pipeline of components that enable end-to-end processing of visual data into motion trajectory adaptations, and which relies on event-based vision and an SNN. Within this pipeline, an event camera emulator transforms RGB data into event data, which is in turn processed by a convolutional SNN. The SNN output is then decoded into avoidance accelerations/velocities by an obstacle avoidance component, which employs a potential fields (PF) approach. A motion control component generates end-effector positions to follow a pre-planned trajectory while adapting the trajectory given the output of the obstacle avoidance component using a dynamic motion primitive (DMP) formulation. These components, collectively referred to as an SNN-based obstacle avoidance module, operate in a closed-loop, online procedure for transforming an input planned trajectory into an adapted version that avoids obstacles through SNN feedback. Figure 1.4 illustrates the stages of the pipeline. By design, the EC emulator is easily substituted by a real event camera. We implement each of these components within the ROS framework to enable synergistic data propagation and efficient parallelization. To the author's knowledge, this approach is unique in addressing manipulator obstacle avoidance by utilizing event data from an on-board camera, SNN processing, and an adaptive trajectory representation.

An integral focus of this thesis is a thorough evaluation of our approach, despite its novelty and the relative scarcity of comparable works, proposed evaluation methodologies, and performance metrics. We therefore formalize a set of quantitative metrics and qualitative criteria with which we can systematically evaluate our implementation. These evaluations are instrumental during the development stage for assessing different parameterizations as well as in the final experiments to draw well-grounded conclusions.

Following the formalization of these criteria, we conduct experiments to evaluate our approach in simulation and on the real robot. These experiments constitute repeated executions in a range of static and dynamic obstacle task scenarios drawn from a defined distribution. During simulation tests, we adopt a machine-learning inspired methodology of using different task scenarios to tune, validate, and finally

test sets of parameter values. The best-performing parameter set is then transferred to the real robot, with minimal adaptations, for the final experiments involving a subset of the task scenarios. The results demonstrate the consistent success of our approach in avoiding obstacles, which a non-adaptive baseline is incapable of. We also analyze the statistical performance across many trials, showing that the adapted trajectories do not drastically increase execution times, trajectory lengths, and velocities, for example, while successfully achieving obstacle avoidance. Moreover, we assess how reliable, predictable, safe, and natural the trajectories are in our qualitative evaluation, and find at least moderately positive results in each. By examining differences between task scenarios, we also present interesting insights concerning different task variables, a notable example being a difficulty with faster dynamic obstacles.

In further tests, we additionally explore certain properties of the neuromorphic elements of our pipeline in isolation. We implement and compare different event emulation strategies, based on their event outputs, SNN responses, and ultimate task performances; this yields the insight that the SNN exhibits favourable robustness to variations in event data. To validate the utility of our SNN component, we test the exclusion of the SNN and the derivation of obstacle avoidance accelerations directly from raw events, which we show adversely affects the success of obstacle avoidance. We also investigate the effect of varying SNN weights by sampling different random values, showing a slight variance in obstacle avoidance performance.

Finally, we show the successful integration of a real event camera, a DAVIS346, in place of the emulation component in our pipeline, and present results of preliminary experiments on the robot. Most notably, we find that the resultant obstacle avoidance performance is fairly similar, although the camera possesses a large number of tunable parameters which merit further experimentation.

This thesis report is structured as follows. Chapter 2 contains the review of related work on the aforementioned areas of research. Chapter 3 provides additional background information on event-based vision, SNNs, and DMPs. This chapter provides supplementary information in case the reader is unfamiliar with these topics, but is otherwise not essential for a reasonable appreciation of the proposed approach. In chapter 4, we present our proposed approach, elaborating on the design principles and each of the components of the pipeline, in addition to details on how this is implemented in software. Chapter 5 is dedicated for a detailed description of the methodology we follow to evaluate our approach, including evaluation tasks, metrics and criteria, and experiment design and procedures for both simulation and real robot experiments. Chapter 6 contains the full results of these experiments and an exhaustive discussion thereof. This chapter also includes the post-experiment analyses concerning the event emulation and SNN components, as well as the preliminary real EC tests. The report ends with a final conclusion presented in chapter 7.

2

Related Work

In this chapter, we conduct a review of research on topics relevant to this thesis, summarizing key insights, investigating methods employed in similar problems, and drawing comparisons to our approach. The objective is to provide self-contained overviews of the areas that the proposed approach lies at the intersection of, in addition to recent examples of their applications that substantiate their advantages and thus motivate their consideration in this work. We also focus on analyzing particularly similar manifestations of the ideas of this thesis near the end of the chapter to highlight differences and perceived limitations, and suggest aspects of our work that may present contributions to this line of research.

We commence with a focus on retina-inspired, event-based vision and event cameras (ECs) in section 2.1 that does not involve spiking neural networks (SNNs), including discussions of their properties, examples of their utility in general computer vision and robotics problems, and emulators. As will become evident, SNNs are often the choice for event-based processing due to their natural compatibility. Section 2.2 is dedicated to SNNs and provides an introduction and a review of publications that examine their computational power, survey the growing field in varying capacities, present and explore different approaches to learning, demonstrate applications in AI and robotics exploiting their unique properties, and contribute neural simulators. Both ECs and SNNs are products and constituents of research into neuromorphic or brain-inspired computing, which is the topic of section 2.3. We briefly examine studies that discuss the merits of mimicking biological computational models derived from neuroscience, and practical implementations on neuromorphic processors[1] that make full use of event data and SNN dynamics, confirming substantial potential in reducing energy consumption and latency. The section closes with a mention of the adjacent field of neurorobotics. Section 2.4 deals with obstacle or collision avoidance methods and focuses on implementations that most closely resemble our own in their application domain, approaches to the problem, and/or sensing modality. We discuss some limitations of said implementations and how ours differs in comparison, before concluding with final remarks.

2.1 Event-Based Vision

Event cameras (ECs), also called silicon retinas, neuromorphic cameras, or dynamic vision sensors, are models of biological retinas, inspired by the latter's superior efficiency in comparison to conventional, frame-based cameras. ECs transmit streams of spatio-temporal "events" when light intensities in independent

[1] For brevity, we henceforth abbreviate the term "neuromorphic processor" to the non-standard "neuro-processor".

"pixels" cross a reference threshold, mimicking the behaviour observed in retinal photoreceptor cells. Each asynchronous event encodes pixel location, change polarity, and emission time point. Therefore, EC event streams selectively capture significant visual stimuli, mostly due to motion, in contrast to frame-based cameras, where pixels synchronously and continuously transmit absolute intensity values. The silicon retina was pioneered by Mahowald and Mead in the 1990s (Mahowald (1994)), and paved the way for modern ECs including the DVS (Lichtsteiner et al. (2008)), ATiS (Posch et al. (2010)), and DAVIS (Brandli et al. (2014))[2], which have fostered research into event-based vision and facilitated demonstrating its advantages. See Gallego et al. (2022) for an extensive overview of the field of event-based vision that provides details exceeding the scope of this section.

Apart from their appeal in reproducing observably superior qualities of biological retinas, ECs have demonstrated practical advantages over frame-based cameras. In their review of neuromorphic cameras, Posch et. al. highlight the deficiencies of frame-based cameras, chief among which is an artificial, clock-driven frame capture rate which is independent of scene dynamics (Posch et al. (2014)). This can lead to under-sampling, where information between consecutive frames is lost, or wasteful data transmission in the absence of inter-frame changes. On the other hand, EC pixels transmit events independent of a global shutter and thus at very low latencies, effectively capturing high-speed dynamics. This higher temporal resolution also mitigates the effects of motion blur, another challenge for normal cameras. (It is worth noting that these challenges are better addressed, not wholly eliminated, by ECs, as stressed in Hu et al. (2021).) Furthermore, ECs have high dynamic range, more than double the average 60dBs of high-end conventional cameras, which equates to much better adaptivity to extreme lighting conditions (as our eyes possess). These benefits are visually observable when reconstructing video from EC data such as in Rebecq et al. (2019), where recurrent neural networks (RNNs) are trained for this purpose and produce images at more than 5000 frames per second with high dynamic range and robustness to motion blur. In addition, the transmission of information only at significant intensity changes inherently eliminates redundant information. The significantly lower rate of data transmission naturally leads to higher energy efficiency and lower power consumption, a particularly prominent characteristic of ECs (most run at about $10mW$, Gallego et al. (2022)). All of these properties make ECs particularly appealing for autonomous systems tasked with any significant visual processing in varied and unpredictable environments. This potential has spurred research into developing and demonstrating methods and algorithms that enable harnessing said properties for general computer vision applications.

Recent publications highlight applications of event-based vision in a variety of domains. A survey of bio-inspired sensing in the field of autonomous driving is presented in G. Chen et al. (2020), providing a review of EC signal processing techniques and successful implementations for relevant tasks including segmentation, recognition, OF estimation, image reconstruction, visual odometry, and drowsiness detection. In a more resource-constrained scenario, event-based vision has also been employed in a gesture recognition smartphone application in Maro et al. (2020). Here, the authors process data from an ATiS EC in a "time surface" representation to learn hierarchies of spatio-temporal features, which are sufficiently expressive for a nearest neighbour classifier to achieve good gesture classification even for visually-impaired users.

[2]DVS: Dynamic Vision Sensor; ATiS: Asynchronous Time-Based Image Sensor; DAVIS: Dynamic and Active-pixel Vision Sensor

An interesting method is employed to detect and dynamically suppress irrelevant background events by leveraging the higher local activities surrounding foreground event pixels. In Dubeau et al. (2020), a novel combination of RGB-D and EC data for DNN-based 6-DOF object tracking is shown to outperform a state-of-the-art RGB-D-only DNN. Two networks are separately trained to estimate a given object's pose: one on an "event spike tensor" representation of DAVIS346 data (trained on ESIM simulator data), the other on RGB-D frames from a Microsoft Azure sensor. Inferences from the former are then used to refine the latter's in a cascaded approach which, while relying on a prior initial pose and a 3D textured model of the object, demonstrates the utility of event data in improving an existing tracking network.

In the context of robotics, ECs have most often been utilized on aerial robots, such as for high-speed obstacle avoidance (Falanga et al. (2020)) powerline tracking (Dietsche et al. (2021)), and fault-tolerant control (Sun et al. (2021)). UAVs particularly benefit from the aforementioned advantages of ECs on account of frequent high-speed motions and motion blur, energy consumption constraints, and rapid changes in illumination, which motivate these works' consideration of event-based vision. We dedicate attention here to applications that do not involve flight, in which ECs may similarly provide opportunities for the improvement of robot capabilities.

Prior to the advent of modern ECs, Bečanović et. al. used precursory optical analog VLSI (aVLSI) sensors in soccer robots, harnessing the faster reactivity properties to actively control a ball and estimate its velocity (Becanovic et al. (2002)), and to improve robot goalkeeping (Bečanović et al. (2002)). Succeeding silicon retinas share similar principles, such as modeling retinal cell dynamics using sub-threshold transistor regimes, but now offer resolutions exceeding 10x10 and spike-like event streams instead of OF estimations. The classical problem of simultaneous localization and mapping (SLAM) has been tackled in a novel application of stereo visual odometry using DAVIS346 ECs in Y. Zhou et al. (2021). On the basis of analyzing spatio-temporal consistencies in dual event streams and measuring distances in "time surface" event representations, the proposed approach facilitates scene mapping and ego-motion estimation, matching the performance of mature frame-based approaches on benchmark datasets while demonstrating robustness in difficult lighting conditions. Others have experimented with event-driven visual frame representations. In Arakawa & Shiba (2020), EC data arranged in a conventional image frame format was used for visual reinforcement learning (RL) of obstacle avoidance and tracking policies. The authors train in simulations using emulated event data and deploy the model on a real robot equipped with a DAVIS240 EC. In H. Chen et al. (2019), similar event representations are used to train DNNs for Atari gameplay and action recognition, outperforming RGB image-based networks. While the latter works adapt DNNs to process event frames, the present work will look into utilizing SNNs, which are naturally suited to processing the spike-like event data.

For this work, we implement a software component that emulates event data from a stream of conventional images in a live camera feed or a ROS topic (see section 4.1.2). This is not a novel idea, and we conclude this brief investigation of the literature on event-based vision with a look at previous implementations of the concept of event emulation/simulation. pydvs is a python-based DVS emulator that can convert image intensity differences to rate- or time-encoded spikes/events, utilizing local inhibitions, max operations, and redundant spike discarding strategies for refining the emulated data (García et al.

(2016)). This closely resembles our emulator, but differs in not providing native ROS support, which is particularly useful in robot applications. The prominent ESIM EC simulator of Rebecq et al. (2018) provides a framework for simulating 3D scenes and user-defined camera motions, providing accurate event outputs, and has been used in various works. The simulator depends on a selection of rendering engines, including OpenGL and the Unreal engine, to simulate scenes. Microsoft's AirSim provides an event camera emulation component[3] used in some UAV-related studies, but which is similarly coupled to its rendering engine. Such a coupling limits applicability to the associated simulation. The v2e toolbox (Hu et al. (2021)) was designed with the purpose of addressing some reported assumptions of the ESIM simulator that deviate from real cameras, including the absence of temporal noise and leak events. The tool emulates DVS events from video data, and its output has been shown to increase robustness to lighting conditions in a car detection task. However, the implementation natively supports only video files, and not real-time camera feeds nor ROS topic data, for example, which are requisites in the present work. In Joubert et al. (2021), the so-called ICNS emulator is introduced for video files and Blender scenes, and qualitatively compared to ESIM, v2e, and a real DVS. All reviewed emulators are publicly available and provide valuable, extensible tools for research and development in event-based vision. Nevertheless, their coupling to rendering engines and the absence of out-of-the-box support for live feed and ROS data processing motivates the development of the event camera emulator described in section 4.1.2.

2.2 Spiking Neural Networks

SNNs represent more faithful models of biological neurons than conventional artificial neural networks (ANNs) do, differing primarily in their information propagation mechanisms. While ANN units propagate real-valued, constant signals, spiking neurons communicate in trains of discrete, sparse pulses or *spikes*, mimicking experimentally-observed neuronal signals in living organisms. Incoming spikes contribute to the decaying membrane potential of a neuron, which fires (generates a spike or *action potential*) only when this quantity crosses a threshold. The time-varying, analog dynamics within each neuron result in an asynchronous propagation of binary signals, not unlike the events described in section 2.1. Research into spike-driven NNs is motivated by attempts to approach the impressive computational capability coupled with energy efficiency of the human brain (as expressed by the majority of works reviewed in this section), compared to the more specialized yet drastically less efficient computer systems of today (Roy et al. (2019)). The spatio-temporal distributions of spikes are thought to efficiently encode information, while their sparsity contributes to lower energy expenditure. The homogeneity of events from ECs and spikes in SNNs make the latter a natural choice for processing data from the former (resembling the processing of retinal information on its way to and within the visual cortex).

The potential of SNNs as the "third generation of neural networks" (the first two being McCulloch-Pitts neurons and neurons with non-linear activation functions, respectively) has often been studied and demonstrated in the past. In a rigorous study on the computational power of SNNs, Maass presents theoretical proofs of SNNs being as, and potentially even more, expressive than first and second-generation NNs, in addition to requiring significantly less neurons to represent some functions (Maass (1997)).

[3] https://microsoft.github.io/AirSim/event_sim/

More recently, researchers have attempted to formulate models that enable systematic analysis of SNN computational power. In Kwisthout & Donselaar (2020), a "neuromorphic complexity theory" is presented which encapsulates complexity classes, completeness theorems, and other elements of a formal SNN machine learning model. These works indicate ongoing and growing efforts at formalizing a theory of SNN computation. The authors of Neil et al. (2016) convert ANNs pre-trained for MNIST digit recognition into SNNs, whose inputs are spike-train representations of the original images, and analyze the spiking versions' comparative performance. The SNNs achieve comparable accuracy using significantly less computational operations (42-58% less) and in less time, since a reliable inference could be made before observing the complete sequences of input spikes. These results demonstrate that SNNs could be as expressive/accurate as ANNs, while consuming less power and exhibiting faster inference.

The expansive body of recent, seminal publications on SNNs is challenging to reasonably cover without approaching the scale of a dedicated survey. Therefore, we constrain this brief overview by firstly highlighting a selection of particularly notable surveys that provide much more complete treatments of the topic. We then briefly review publications that exemplify the varied approaches to learning in SNNs, and others that present notable SNN applications, applications in the robotics domain, and neural simulators.

The recent prevalence of deep learning with ANNs has naturally ignited research into the same in the spiking domain, not least because of prospective improvements in energy efficiency, a characteristic challenge with conventional deep neural networks (DNNs). Pfeiffer & Pfeil (2018) and Tavanaei et al. (2019) both provide extensive overviews of SNNs and focus on methods for training deep SNNs, drawing comparisons to conventional DNNs, and reviewing spiking analogs of CNNs, RNNs, LSTMs[4], and echo state networks, among others. The authors of Bouvier et al. (2019) survey the literature with a focus on hardware implementations that leverage SNN characteristics (including neuro-processors, discussed in section 2.3) and the associated challenges, additionally reviewing approaches to SNN learning. In Jang et al. (2019), a probabilistic view of SNNs is presented, the main advantage of which is the differentiability of proposed spike distribution representations that facilitate gradient-based learning and other well-known statistical methods. The authors review various learning algorithms and successful applications of the stochastic spiking model.

As it stands, the majority of research in this field appears to be dedicated to formulating SNN learning algorithms; due to the novelty of spike-based representations, an evident lack of consensus exists on how best to train SNNs (as expressed by prominent researchers in recent discourse: Zenke et al. (2021)). Taherkhani et al. (2020) reviews biologically plausible learning rules for SNNs. Spike-timing-dependent plasticity (STDP), a leading candidate, is a variant of Hebbian learning in which local synaptic connections are strengthened or weakened in proportion to the relative timing of pre-synaptic and post-synaptic spikes, which may encode causal relations. STDP has been primarily used in unsupervised learning for spatio-temporal pattern recognition. Several supervised learning rules have also been proposed until the comprehensive review in Wang et al. (2020), and since (as some of the publications presenting recent applications show). Naturally, most efforts involve attempts at replicating the success of the back-propagation algorithm, but are faced with the challenge of the non-differentiability of a sequence

[4] CNNs: Convolutional Neural Networks; RNNs: Recurrent Neural Networks; LSTM: Long Short-Term Memory

of discrete spikes, as opposed to the familiar real-valued signals. Contributing novel solutions to solve this dilemma has become the focus of much research, with many presented methods seeking to synthesize learning signals from membrane potentials, spike times, and surrogate gradients (which are exclusively studied in Neftci et al. (2019)). Biologically-plausible reinforcement learning has also been demonstrated using reward-modulated STDP (R-STDP), where the aforementioned STDP rule is augmented with a global reward signal (mimicking dopamine) to effect reward-driven or novelty-driven learning. This mode of learning is reviewed in Frémaux & Gerstner (2016). Finally, many approaches involve converting trained ANN models to SNNs, with the objective of reaping the benefits of spike-driven communication, including energy efficiency, while minimizing performance loss (for an example, see Blouw & Eliasmith (2020)).

This research has produced implementations of SNNs for solving common problems in AI, particularly involving vision. Diehl et. al. trained two-layer SNNs with STDP for MNIST digit recognition and achieved state-of-the-art classification accuracy among unsupervised methods (95%) (Diehl & Cook (2015)). This approach does not require labels during training: following the presentation of all training inputs, output neurons are assigned classes according to inputs for which they selectively spike at the highest rate. In Mirsadeghi et al. (2021), a recent supervised learning algorithm that utilizes a temporal instead of a rate encoding of spikes, STiDi-BP, is shown to achieve an accuracy of 99.2% on MNIST. Here, a backpropagation algorithm is enabled by approximating derivatives with the firing times of neurons and calculating errors in the output layer between actual and desired firing times, which are selected such that a particular output neuron consistently fires earliest for each class. For less trivial object recognition, Spike-YOLO was created by converting a pre-trained Tiny-YOLO model to an SNN, achieving comparable results on the PASCAL and COCO datasets, while being 2000 times more energy efficient (when quantified by the number of 32-bit float and integer operations) (Kim et al. (2020)). Similarly, the authors of Zheng et al. (2020) present a supervised learning algorithm, STBP-tdBN, for direct supervised training of SNNs, achieving state-of-the-art results on the challenging CIFAR and ImageNet datasets. This work is among the few to successfully train relatively deep SNNs (of 50 layers or more). Zhou et. al. demonstrate object recognition from datasets of DVS (N-MNIST, DVS-CIFAR10, etc.) and LIDAR (KiTTi) data, demonstrating the applicability of SNNs to different data modalities (S. Zhou et al. (2021)).

Various works have also successfully applied SNNs in the field of robotics. An extensive survey of SNNs in the context of robot control is presented in Bing, Meschede, Röhrbein, et al. (2018), underscoring significant potential for improving speeds, energy efficiency, and computational capabilities for robotics applications. In Bing, Meschede, Huang, et al. (2018), SNNs are trained with R-STDP on DVS event data to accomplish lane-keeping on a mobile robot, outperforming a conventional Braitenberg controller in experimental evaluations. Zahra et. al. utilize shallow SNNs to develop a differential sensorimotor mapping for a UR3 robot that supports reliable Cartesian control and is learned through a motor babbling-like procedure using STDP (Zahra et al. (2021)). In one of the first fully-embedded applications of SNNs (on an Intel Loihi neuro-processor), Dupeyroux et. al. design a neuromorphic vertical thrust controller for landing a quadrotor (Dupeyroux et al. (2021)). The input to a 20-10-5-layer SNN is a spike representation of OF divergence that is estimated using data from an on-board "CMOS camera" (a "position coding", where the input value determines which of the input neurons exclusively fires). The SNN is trained and

evaluated using a neural simulator, PySNN, in simulations, then on a neuromorphic chip in real-world experiments, achieving consistent landing behaviour, and demonstrating robust "sim-to-real" transfer. In the present work, we similarly utilize a neural simulation tool and test in simulations before transferring to a real robot. Instead of implementing any of the previously discussed approaches to learning, the SNN parameters were optimized using an evolutionary algorithm, indicating prospects of applying traditional machine learning methods in the domain of SNNs. A limitation on the generality of this approach is the controller's dependence on a pre-set visual pattern for OF estimation.

In a similar direction, an event-driven, SNN-based PD thrust controller was designed to achieve high-speed orientation adjustment on a dual copter in Vitale et al. (2021). Using the task of matching the orientation of a horizon-like line on a rotating disc in front of the flying UAV, the authors demonstrate superior control speeds and reductions in latencies, particularly when running the SNN on a Loihi neuro-processor instead of a CPU. In addition, the on-chip learning capabilities of the Loihi are utilized in adjusting controller gains (synaptic weights) to adapt to changes in injected steady-state errors, using a simple STDP-like learning rule. A more recent application of ECs and SNNs in a common robotics problem involves a neuromorphic approach to stereo vision. In Risi et al. (2020), reliable stereo matching of event data from two DAVIS sensors is achieved using an SNN architecture designed with neuronal populations implementing coincidence and disparity detectors, and running on a DYNAP neuro-processor. The neuromorphic approach is particularly favoured for the temporal dimension of asynchronous event data and SNN spike data, which enables exploiting temporal coincidences and thus improve stereo matching. Similar to our current approach, no learning is involved; the SNN architecture's inherent properties are shown to be beneficial in realizing the desired behaviour. In general, most publications in robotics address relatively constrained navigation and flight tasks (for e.g. lane-keeping and one-dimensional thrust control). This thesis aims to demonstrate a less common application in obstacle avoidance for manipulation.

We close this section with a brief overview of work on SNN simulators: software tools and packages that enable creating and running SNNs outside of dedicated neuro-processors. Many publications referred to in this report have used one to obtain cited results. The NEST (neural simulation tool, Gewaltig & Diesmann (2007)) and BRIAN (Goodman & Brette (2008)) packages both enable constructing arbitrary neural architectures and defining neuronal dynamics through differential equations. The NeMo platform (Fidjeland et al. (2009)) provides tools for designing networks of Izhikevich neurons[5] at a higher level of abstraction and accelerating simulations with GPU parallelization. Neucube (Kasabov (2014)), a MATLAB-based framework, enables similar high-level modeling of SNNs with the objective of supporting accurate models of the brain. Unlike others in this list, it is not publicly available. Nengo (Bekolay et al. (2014)) is a toolkit for simulating large-scale SNNs (used to implement "the most complex simulated brain" model: SPAUN[6]), and has primarily been utilized in converting pre-trained ANNs to SNNs in various works (Blouw & Eliasmith (2020), Salvatore et al. (2020), Reiter et al. (2020)). The CARLsim library (Beyeler et al. (2015)) provides GPU-accelerated tools for simulating "biologically detailed" SNN models. Some of the aforementioned libraries support interfacing with neuro-processors such as the Intel Loihi. The PyNN python library offers a simulator-independent framework for specifying SNN models,

[5]Refer to section 3.2.1 for a description of the Izhikevich neuron model.
[6]Semantic Pointer Architecture Unified Network

which can run directly on supported simulators (for e.g. NEST, BRIAN, CARLsim, and NeMo) and neuro-processors (for e.g. SpiNNaker and BrainScaleS) (Davison et al. (2009)). The BindsNET library provides a Pytorch-based implementation of SNNs, geared towards machine learning and reinforcement learning applications (also integrating with OpenAI Gym environments). The library is presented in Hazan et al. (2018), where the authors also compare their implementation to other neural simulators'. Apart from pre-packaged neuron models, synaptic connection types, and learning rules, the package is designed to facilitate extensions and custom implementations of each. BindsNET is used in the present work, primarily for this extensibility and its direct interface with existing PyTorch functionalities.

2.3 Neuromorphic Computing

Having surveyed neuromorphic approaches utilizing ECs and SNNs, we briefly discuss the concepts of *neuromorphic computing* as the central idea initiating research on both, neuromorphic processors that better leverage SNN capabilities, and practical implementations demonstrating the concrete advantages thereof.

Neuromorphic computing/engineering[7] endeavours to mimic the fundamental neural architectures and dynamics of the biological brain *in silico*, aiming to replicate its characteristically superior energy efficiency, compute, and robust learning capabilities in modern computer architectures and engineered systems, in both algorithms and hardware. Common approaches primarily incorporate brain-inspired principals that include asynchronous event-driven communication, spike-based neural processing, analog neuronal dynamics, and local synaptic adjustments. Although results constitute more biologically-plausible systems (than, for example, conventional DNNs), the goal is often to reproduce functionalities at adequate levels of fidelity and abstraction that nevertheless exclude unnecessary details (Wunderlich et al. (2019)). Challenges include open questions in neuroscience, such as how a mode of communication that essentially relies on ionic flow in salt water, which in theory would be much slower than communication in silicon-based systems, leads to primitive organisms like the housefly outperforming today's autonomous, AI-powered vehicles (Mead (2020)). Neuromorphic computing thus serves a dual purpose: enhancing AI systems with lessons from neuroscientific research, and advancing our understanding of the brain by experimenting with neurologically-inspired platforms. Among the most prominent results of this research are ECs (discussed in section 2.1) and neuro-processors designed to run the novel architectures of SNNs. A review of neuromorphic hardware and practical applications can be found in Rajendran et al. (2019), which highlights the role of neuro-processors in realizing the full potential of SNNs for exploiting event-based sensing, learning and inference.

The establishment of the neuromorphic computing field is credited to Carver Mead, who proposed studying biological systems' more effective forms of computation to improve computing architectures

[7] Technically, the terms neuromorphic engineering and neuromorphic computing are not equivalent. The first predominantly concerns the design of electronic components that can replicate observed neural dynamics (such as transistors operating in sub-threshold regimes or memristors), while the second focuses on developing computing architectures and principles that depart from the von Neumann model and approach neural models such as SNNs. However, they have been used synergistically, if not interchangeably, in the literature in reference to the general concept of mimicking the intelligence and efficiency produced by the nervous system through more biologically-plausible design. We similarly do not stress the distinction here, and use *neuromorphic computing* to refer to this concept throughout this report.

and address their foreseeable limitations (Mead (1990)). As suggested more than three decades ago, an example is conventional computers scaling up in computational power while not scaling down as favourably in computational costs and thus remaining far less efficient than the human brain, as the trend predicted by Moore's law[8] is halted by fundamental physical limits in transistor design (including effects such as quantum tunneling). The proposal is thus to explore drastically different computational paradigms, such as neuromorphic computing, that may aid in escaping this inevitability. The conventional von Neumann architecture is also limited by the traditional segregation of processing and memory units; computations are hampered by the transfer of information between the two at limited bandwidths, leading to losses in energy and time (Indiveri & Liu (2015)). This problem, often termed "the von Neumann bottleneck", can be addressed by the co-location of memory and processing as implemented in modern neuro-processors instead of a separate, monolithic memory unit (Rajendran et al. (2019)). Such a design moves away from the mostly serial von Neumann information processing and towards the massively parallelized computations of the brain, contributing to reductions in power consumption and latency. This potential of neuromorphic processing, while still largely theoretical, could be beneficial in various application domains, including robotics.

Research into realizing the aforementioned potential has been fostered by the development of various CMOS-based neuro-processors since the turn of the century. These processors enable running SNNs by often modeling membrane potential evolution using voltages across capacitors or transistor sub- or supra-threshold dynamics, and transferring spikes via an address event representation. Furber (2016) provides a survey of pioneering neuro-processors, namely IBM's TrueNorth, Stanford Neurogrid, BrainScaleS and SpiNNaker (the last two originating from the Human Brain Project), and compares their performances across various dimensions (energy efficiency, programmability, speed, etc.). Other notable surveys describe and provide similar statistical comparisons of these processors, in addition to the more recent Intel Loihi, DYNAP, PARCA, Braindrop, ODIN, and Deepsouth (Bouvier et al. (2019), Thakur et al. (2018), Rajendran et al. (2019)). The Intel Loihi and its recent applications have been extensively discussed in Davies et al. (2021), which outlines empirical improvements over von Neumann processors and current limitations of the Loihi and neuro-processors in general. The variety of domains the Loihi was shown to be successfully implemented in demonstrates the general applicability of SNNs, while the quantified gains in energy efficiency validate the benefits of the neuro-processor. In addition, the authors find that conventional DNNs exhibit little to no benefits when run on the Loihi, but SNNs achieve orders of magnitude less energy consumption and latency in some applications. This appears to be in part due to the spiking neuron dynamics simulated within the Loihi contributing to SNN computational capabilities but not being utilized by DNNs, rather only incurring some additional costs in energy and time. These results highlight the utility of the spiking computation model, since attempts at forcing conventional algorithms and models on the neuro-processor may nullify the supposed benefits. While much research has been dedicated to the aforementioned neuro-processors, the recent releases of novel neuro-processor designs (such as SPOON; Frenkel et al. (2020), and μBrain; Stuijt et al. (2021)) indicates continuing

[8]The observation and prediction by Gordon Moore that the number of transistors in an integrated chip doubles every two years, thanks to advancements in transistor design. This corresponds to an increase in computational power on chips of the same size.

efforts in developing innovative neuromorphic architectures. An adjacent line of research pursues a class of memristor-based processors that model dynamics differently, where the variable conductances/resistances of memristors could encode information and model adjustable synaptic connections, for example.

Several more publications investigate the effects on speed and energy efficiency when algorithms are run on neuro-processors. In a practical study to test the advantages of neuromorphic computation, SNNs trained with R-STDP on a BrainScaleS2 neuro-processor were used to control an agent in playing the game of Pong (Wunderlich et al. (2019)). The authors observed one and three orders of magnitude improvements in speed and energy efficiency (through current measurements), respectively, when the SNN was run on the neuro-processor compared to a NEST-based CPU simulation. The authors of Ceolini et al. (2020) address the problem of hand-gesture recognition with a neuromorphic sensor fusion approach, where DVS streams and EMG signals (converted to spikes/events) are used to train SNNs running on a Loihi or ODIN neuro-processor through the spike-based SLAYER backpropagation algorithm. With respect to a GPU-based implementation, the neuromorphic alternative is reportedly at least 30 times more energy-efficient, though inference is 20% slower. Here, energy efficiency is quantified by the energy-delay product (EDP): a product of average energy consumption and inference time. Taunyazov et al. (2020) present a visual-tactile SNN (VT-SNN) which fuses data from a Prophesee event camera and a novel spike-based tactile sensor to accomplish robot manipulation tasks requiring object classification and slip detection. In comparison to state-of-the-art DNNs run on GPUs, the SNN classifiers running on a Loihi performed similarly while consuming 1900 times less power and exhibiting lower latency (since reliable inferences could be made before all spikes from the full input had been processed). In the domain of speech recognition, Blouw et. al. convert pre-trained DNNs to SNNs, then compare their performances when run on an "ultra low power" Movidius Neural Compute Stick and a Loihi neuro-processor, respectively (Blouw & Eliasmith (2020)). Results show a four-fold increase in energy efficiency for the converted SNNs compared to the DNNs. These gains are believed to be due to the amount of "synaptic operations" (SOPs) in the SNNs being proportional to actual activity in the input, unlike the FLOPs[9] in the DNNs, and the ability to substitute the relatively expensive dot-product operations with sums of binary values. In Göltz et al. (2021), a proposed backpropagation rule is used to train SNNs for MNIST classification on a BrainScaleS processor and then compared to the performance of a conventional CNN running on a Nvidia Tesla P100 GPU. Using an energy-per-classification metric, the authors show that the neuromorphic implementation is (approx. 100 times) more energy-efficient, at the cost of a slight drop in accuracy and the number of classifications per second (since the GPU implementation utilizes parallelization, while individual images must be processed sequentially on the SNN). The advantages of neuro-processors in energy efficiency are particularly evident, though further research on SNN architectures and learning may aid in eliminating any deficiencies in performance.

A particularly interesting, related field of research is that of neurorobotics, which involves the design of computational structures that are inspired by the human and animal nervous systems in robots (Van Der Smagt et al. (2016)). Just as neuromorphic computing studies brain-inspired computer architectures, the field of neurorobotics studies brain-inspired robot sensing and actuation. This encompasses replicating

[9]Floating point operations per second

distinct brain regions and their connectivity patterns, with inspiration mainly drawn from the field of neuroethology: the study of the neural mechanisms underlying animal behavior, especially the sensorimotor loops involved in executing complex behaviors. In a synergistic relationship (as in the case of neuromorphic computing), neurorobotics is guided by neuroscientific findings in developing novel robot designs, which in turn can provide useful test beds for furthering our understanding of the brain and nervous system.

A review of the field in K. Chen et al. (2020) demonstrates the utility of neurorobotics for explaining how neural activity gives rise to intelligence, as a form of "computational neuroethology". The authors survey a variety of neurorobotics applications that have been used to study aspects of biological behaviour, including perception, navigation, memory, attention, locomotion, and social interaction. Particularly interesting implementations, referred to as "neuromorphic robots", incorporate neuromorphic hardware (TrueNorth neuro-processor, DVSs, etc.). Among examples of robot designs with high levels of neurological inspiration is shown in Dumesnil et al. (2016), where SNNs are used to implement classical and operant conditioning through STDP in a maze navigation task. The two biologically fundamental modes of learning are realized in a neurologically-grounded design of a mobile robot platform, composed of a brain (a FPGA module), a sensory system (an RGB camera and proximity sensors), a nervous system (an Arduino microcontroller), and a body. The authors of Lobov et al. (2020) present a similar approach to classical and operant conditioning with STDP-based SNN learning on a LEGO robot for simple obstacle avoidance during navigation. Through self-organization of neural pathways, the robot learns associations between certain sonar readings (conditional stimuli) and tactile sensor readings (unconditional stimuli) following collision events, which then help in avoiding future collisions. After learning, the robot's sense of touch is removed, and it is able to predict and anticipate collisions only from sensor readings, due to the learned associations. The Neurorobotics Platform (NRP) provides a framework for the design and experimentation of neurorobotic systems in simulated environments, allowing the design of robot bodies and brain models or "neuro-controllers", connecting them to sensors and actuators, and running experiments like these in simulation (Falotico et al. (2017)). Similar to general computer and engineering, robotics could benefit from ongoing research and development efforts aimed at drawing inspiration from the brain.

In the present work, we do not run SNNs on neuromorphic hardware, instead aiming to investigate the feasibility and utility of an event-based SNN approach for the chosen robotics problem on conventional hardware. Nevertheless, the reviewed research motivates such a study on the basis of exploiting the potential gains in energy efficiency and latency, as the subsequent step; a successful realization of the proposed approach would motivate incorporating neuromorphic hardware in future extensions. On a broader scale, this may additionally inspire the pursuit of neuroethology-based robot designs that advance further into the realm of biological realism.

2.4 Obstacle/Collision Avoidance

Obstacle avoidance is a critical feature for planning robot motions in evolving, dynamic environments, where obstacles to task completion may appear in real-time and invalidate initial motion plans. This section encompasses discussions on particularly seminal works in the domain of collision/obstacle avoidance,

paying special attention to those employing methods relevant to this thesis, and further on approaches tackling manipulation problems and utilizing camera sensors.

Research on the rudimentary issue of reactively computing collision-free paths has lead to a long history of established methods, such as vector field histograms (VFH) (Borenstein et al. (1991)), the Dynamic Window Approach (DWA) (Fox et al. (1997)), and the elastic strips framework (Brock & Khatib (2002)), to name a few. Among the first was Khatib's artificial potential fields (PF) method, which represents task criteria in the form of attractive and repulsive forces acting on an agent moving within a virtual force field (Khatib (1986)). PF techniques have been extensively applied and improved upon over the years, and are utilized in the present work. Various categorizations of obstacle avoidance approaches have been presented in the literature, including local vs. global methods (Rai et al. (2014), Zhang et al. (2019)), classes of grid-based, potential field-based, sample-based, and discrete optimization methods (Feng et al. (2020)), among others. A complete survey is not attempted here, but Minguez et al. (2016) provides a helpful review of classical methods.

Optical flow (OF) estimation, a bio-inspired computer vision technique that is applicable to robot obstacle avoidance, shares similarities to methods proposed in this thesis, and thus deserves mention. OF quantifies the apparent motion of light intensity patterns observed on a sensor (biological or synthetic) as it moves relative to observable objects, and is used by organisms, such as honeybees, for navigation (Van Der Smagt et al. (2016)). This can provide estimates of ego- or object motion, which facilitate tracking and collision avoidance, as demonstrated in various works. For example, in Schaub et al. (2016), OF is computed from monocular camera data on an autonomous car and used in an optimizer to derive maneuvers that avoid obstacles. OF estimation approaches are often split into two categories with distinct downsides. In estimating object motion, dense OF methods track changes to every pixel between consecutive frames and sparse OF relies on tracking identified features between frames. The former imposes high computational and memory costs, while the latter depends on the reliability of feature matching algorithms and may not generalize if object models have to be specified a priori (H. Lee et al. (2021)). In comparison, the proposed approach is envisioned to eliminate unnecessary computations by virtue of event-based processing, and be independent of specific obstacle features.

Compared to 3D sensors, IMUs, and laser sensors, cameras are less frequently utilized for obstacle avoidance. Nevertheless, works that do employ monocular cameras offer more relevant comparisons to ours, though most concern mobile robots and UAVs. H. Lee et al. (2021) demonstrates DNN-based obstacle recognition and avoidance on a UAV navigating a plantation, where trees are recognized, distances are estimated, and free regions in the image space are determined for simple heading adjustments. However, limitations include a restriction to obstacles that the DNN is trained to recognize and the computational expense of running the model. In Hua et al. (2019), semantic segmentation DNNs are trained to recognize roads and obstacles a mobile robot encounters, which are then incorporated in PF-based local path planning. More rudimentary approaches involve classical computer vision methods on RGB data, such as detecting contours for obstacle detection (Martins et al. (2018)) or using feature extractors like SURF to recognize known obstacles (Aguilar et al. (2017)), followed by searching for free regions and applying corrective motions. These approaches may be susceptible to changes in lighting conditions, where ECs

are expected to perform better (see section 2.1). In addition, local heading adjustments characteristic of purely reactive approaches necessitate subsequent computations of rectifying velocity commands that return the agent to its original path. In our work, we propose using dynamic motion primitives (DMPs) for adaptive high-level path plans that obviate the need for extra corrective computations. The significant on-board computations associated with some of these works may also present opportunities to demonstrate neuromorphic solutions that could impose less.

With regards to robot manipulation, obstacle detection and avoidance could be a particularly crucial capability in safety-critical human-robot collaboration (HRC) settings. The authors of Chiriatti et al. (2021) design a control law for a UR5 manipulator that incorporates collision cylinders instantiated from estimates of obstacle geometries, positions and velocities, demonstrating avoidance behaviours with different constraints placed on arm motion. The method was tested only in simulation with prior obstacle information; extending this to real scenarios would require dedicated sensors and algorithms for estimating the pose of every person and object with reasonable accuracy. In Safeea et al. (2019), collision bounds on a person are incorporated in a PF approach to controlling a KUKA LBR iiwa arm. These bounds are obtained using IMUs placed on persons in an industrial workspace. A recently proposed approach to manipulator collision management relies on novel proximity sensors placed on the arm, which provide time-of-flight, IMU, and gyroscope readings (Escobedo et al. (2021)). Using quadratic programming (QP) for motion control, the authors achieve not only avoidance but also anticipation and post-contact trajectory adaptation. While impressive and demonstrably reliable avoidance behaviours are achievable by deploying arrays of sensors in the environment, objects/persons, or the robot itself, this may limit generalization to different scenarios, especially when a robot is not confined to a controlled workspace or when arbitrary sensor placement is not possible. In contrast, our approach does not rely on external sensors placed on objects or in the workspace, in favour of a single on-board camera.

Other notable implementations in manipulation scenarios integrate vision-based sensing, leveraging RGB-D cameras in particular. Mronga et. al. use pointcloud data to extract convex hulls of obstacles and persons incorporated as constraints in an optimization problem whose solution leads to avoidance motions on a KUKA LBR dual-arm system (Mronga et al. (2020)). The optimizer leads to task-compliant obstacle avoidance, but depends on multiple RGB-D cameras covering the workspace, and several pointcloud processing steps. This is similar to a strategy followed in Song et al. (2019), where obstacles interfering in a bin-picking task are avoided by detecting moving objects in a pointcloud and accordingly adjusting motions of a Techman TM5-700 arm through a PF algorithm. Here, data from two dedicated Kinect cameras, placed in appropriate positions in the workspace, are pre-processed to remove known background elements and the robot arm, thus limiting the approach to the environment it was designed for. These implementations benefit from inherent depth perception that monocular RGB or event camera-based methods do not. However, event-based processing is expected to impose a significantly lower computational overhead than pointcloud processing, which may be a particular concern in applications that require rapid reactivity.

A few publications are particularly relevant to this thesis for their similar approaches to obstacle avoidance, and thus merit special consideration in the remainder of this chapter. We start with examples

of online trajectory adaptation that utilize PFs. In Park et al. (2008), an obstacle avoidance term in the equations of dynamic motion primitives (DMPs) is first introduced and used within a PF formulation to enable avoidance of static and dynamic obstacles in simulation and on a real manipulator. We similarly utilize DMPs here as online-adaptive motion planners, and draw insights from the authors' mathematical integration of obstacle avoidance information which informs part of the approach presented in section 4.1. In a similar direction, Scoccia et. al. present an approach to offline planning of manipulation trajectories along with online adjustments for obstacle avoidance, using a formulation of potential fields that operates on the Jacobian matrices for null space control and evaluating in simulations (Scoccia et al. (2021)). Another recent implementation of manipulator obstacle avoidance from Oussama Khatib's group combines PFs and elastic bands for adaptive trajectory planning on a Franka arm (Tulbure & Khatib (2020)). As in aforementioned works, an RGB-D camera capturing the workspace provides pointcloud data which is processed to estimate obstacle positions that affect the PF. The authors augment a PF algorithm with an elastic bands planner, which enables adjusting a global plan with minimum deviations, thus addressing the susceptibility of PFs to local minima solutions. This resembles our application of PFs for local velocity corrections, while DMPs maintain an adaptive high-level plan that ensures global convergence to the goal. In similarity to related works, pointcloud processing introduces a computational expense (the bottleneck of the approach, as expressed by the authors) and a dependence of pre-processing procedures on the camera position and/or robot platform, thus placing theoretical limits on generality and applicability to different environments. Both problems may be tackled by the on-board event camera proposed in the present work. Nevertheless, these approaches share a central idea with the present thesis: online adaptation of pre-planned trajectories to address unpredictable environment dynamics in real-time.

An integral aspect of our work is event-based vision (to which section 2.1 is dedicated), whose utility for obstacle avoidance has been demonstrated in several publications. Milde et al. (2015) presents a preliminary application of event-based collision avoidance on a mobile robot by computing OF from DVS data and deriving simple velocity commands. Here, the usage of ECs is motivated by the redundancy in data and wastage of computations associated with processing conventional camera images, particularly when the robot is stationary. The authors suggest extending their work with a "neuromorphic circuit" and SNNs to address some limitations, including a PCA-based method for computing OF from events that requires a significant amount of data. In Sanket et al. (2020), dynamic obstacle avoidance on a quadrotor is achieved by training CNNs on event frame data to estimate the OF of moving objects in a semi-supervised fashion, while placing priors on obstacle shape (sphere) for tractability. The computed flow is then used in a PID controller for avoiding flying objects while maintaining a reference position. This differs from the type of tasks we seek to address, where the robot must avoid obstacles whilst actively moving towards a goal. The authors also design an encoder-decoder CNN for "deblurring" constructed event frames for better OF estimates. In the present work, we employ SNNs, instead of DNNs, since they are inherently better-suited to event data processing. In a navigation scenario, Yasin et. al. utilize a DVS for car obstacle avoidance in low-light settings, demonstrating superior reaction times when compared to standard cameras (Yasin et al. (2020)). Objects in the event image are obtained through denoising

(KNN[10]), corner detection (LC-Harris), segmentation (hough transforms) and filtering procedures, and used in a novel re-planning algorithm. While these efforts present viable applications of event data, much of the requisite pre-processing and processing may be obviated by utilizing SNNs: the natural complement to event-based vision, as is pursued in the present work.

This idea of SNNs processing event data has nevertheless also been demonstrated in recent years. Salvatore et al. (2020) demonstrates neuro-inspired UAV collision avoidance by running event data on an SNN converted from a trained deep Q-Learning (DQN) ANN. Successful avoidance behaviour is achieved in AirSim simulations after training the DQN agent on emulated event data, transferring learned weights to an equivalent SNN, and further training the SNN with data from successful trials. A similar aspect to our work is the emulation of event data from frame-based camera data to demonstrate the efficacy of the proposed approach. The use of SNNs for processing the spatio-temporal event data is motivated by their natural compatibility. In addition, this strategy for capturing temporal aspects of the task is preferred over LSTM-based modules, which would reportedly impose a heavier computational burden. Perhaps of greatest similarity to our work is the feasibility study of a neuromorphic approach to obstacle avoidance presented in Milde et al. (2017). Their method involves processing event data from a DVS mounted on a mobile robot in SNNs implemented on a ROLLS neuro-processor, whose output is decoded into avoidance and target-following behaviours by aggregating responses of spiking neuron populations. As in our approach, SNN connections are non-plastic (i.e. not adjusted through learning); the inherent properties of the SNN architecture is shown to facilitate viable navigation behaviours. Some of the reported challenges also apply to our work, including i) difficulties reacting to obstacles that suddenly enter the FOV, ii) smooth surfaces producing few events, leading to limited visual saliency, and iii) event noise filtering possibly exacerbating the perception of low-contrast objects, which would produce even less events. The present work aims to study a similar approach but in the context of manipulation, which presents more challenges compared to navigation.

Evidently, a significant amount of recent research on obstacle avoidance deals with UAVs, followed by mobile robots. A relatively small segment is in the manipulation domain, and a few works take a neuromorphic approach that involves either event-based vision or SNNs. To the best of our knowledge, the approach proposed in this thesis is unique in addressing manipulator obstacle avoidance using event data from an on-board camera, SNN processing, and adaptive trajectory representations. Most reviewed solutions rely on external cameras and classical computer vision methods for filtering images, removing background and robot, corner and line detection for object segmentation, etc. Our proposed approach may support the argument that utilizing event data and SNN processing could eliminate the necessity of manual operations that preclude generalization over different environments, lighting conditions, platforms, and sensor setups. To the detriment of comparative evaluations to this approach, most related and compelling works use RGB-D or point cloud data, and fewer exclusively utilize RGB images: the closest analog to event data. This is compounded by the relative scarcity in quantitative metrics and results of obstacle avoidance performance in the reviewed literature, which complicates establishing the current state of the art and grounding metrics on which to objectively evaluate the proposed implementation.

[10]The K nearest neighbours algorithm

Nevertheless, we formulate a list of performance metrics by drawing inspiration from publications that do report quantitative results and suggesting our own (see section 5.1.3) in order to conduct a systematic analysis of performance.

2.5 Concluding Remarks

The review of the literature presented in this chapter has provided notable insights on the central topics of this thesis and a perspective on similar approaches. The stated advantages of event cameras for visually-guided autonomous systems were substantiated by studies on applications in a variety of contexts. Spiking neural networks were presented as a natural paradigm for processing event data by mirroring visual cortical processing, and found to be at least as powerful as conventional networks, compatible with different data modalities, and applicable to many problems, though most publications address limited navigation and flight scenarios. The best method(s) to train SNNs remains a topic of intensive research, but various studies have demonstrated success with promising candidates, while others have even shown the inherent algorithmic utility of SNNs devoid of any learning capabilities. The section on neuromorphic computing has exhibited implementations of event-based vision and spiking networks on neuromorphic hardware, designed to exploit spike-driven communication, and the ongoing development of neuro-processors, providing concrete demonstrations of the potential of brain-inspired computing. While implementations on conventional hardware do not fully realize reported improvements in energy efficiency and communication latency, studies such as the one conducted in this thesis investigate and demonstrate the applicability of a neuromorphic approach, which can then be extended to utilize neuromorphic hardware. Existing software simulators of both event cameras and SNNs, surveyed in the respective sections, provide opportunities for the rapid development and testing of neuromorphic pipelines, as is done in the present work. For the ubiquitous problem of obstacle avoidance in robotics, we find that most efforts have targeted aerial and mobile robots, though a few provide encouraging demonstrations of similar neuromorphic methods. Our novel approach to obstacle avoidance in manipulation may be a step towards addressing limitations of the comparatively high computational expense of conventional RGB and depth data processing, dependence on classical vision methods that hinder generalization and robustness to varying operating conditions, and non-adaptive motion representations in goal-directed tasks, all while contributing to the exploration of the potential of brain-inspired design.

3

Background

In this chapter, we provide extra background information on the topics of event-based vision, spiking neural networks, and dynamic motion primitives. These sections serve as supplementary sources for forming a better understanding of underlying concepts, particularly those that have not been thoroughly discussed in chapter 2. This information may be helpful in particular to the unfamiliar reader, but is nevertheless not vital for understanding the key elements of our approach and subsequent discussions and analyses.

3.1 Event-Based Vision

Silicon retinas were developed and studied by Mahowald and Mead (Mahowald (1994)) with the aim of imitating the neural architectures of biological retinas in analog VLSI circuits[1]. This research gave rise to the field of neuromorphic computing, which investigates the potential in mimicking neurobiological structures *in silico* for progressing the capabilities of AI systems towards the superior efficiency and efficacy of their biological counterparts. Contemporary realizations of these sensors are known as *neuromorphic retinas*, *dynamic vision sensors (DVS)*, or *event cameras (ECs)*, and share the same working principles. The pixels in an EC mimic retinal ganglion cells by asynchronously emitting a binary signal, i.e. an event, only when the incident light intensity significantly changes. Using a so-called address event representation (AER), the pixel arrays provide streams of spatio-temporally-registered events which encode typically interesting information, such as motion. Consequently, ECs selectively acquire information depending on scene dynamics. By contrast, conventional frame-based cameras synchronously transmit absolute intensities at all pixels, driven by an independent external clock. Though familiar and convenient, this information representation can be redundant and/or wasteful.

The practical advantages of ECs are manifold:

- Addressing potential "undersampling" due to fixed frame-rates, where inter-frame information is lost (G. Chen et al. (2020))
- High energy efficiency and low power consumption, with ratings around 10-30mW (Dubeau et al. (2020)) and 1-10mW (Maro et al. (2020))
- High dynamic ranges, facilitating better adaptivity to significant changes in luminosity

[1] Very large-scale integrated circuits

- Low latencies and high temporal resolutions, leading to high effective data capture rates
- Mitigating effects of motion blur

These properties are primarily products of the ECs' novel data representation and acquisition mechanism, which result in capturing the most useful information, often in relative signal changes, and avoiding redundancies. The exploitation of said characteristics for visual processing applications has motivated efforts into developing event cameras and research into event-based vision. Refer to section 2.1 for a further discussion on these properties and for demonstrations in applied research.

These relatively novel sensors are not devoid of drawbacks that limit their wide applicability. Current ECs have relatively low resolutions, such as 128×128 (Lichtsteiner et al. (2008)), 240×180 (Brandli et al. (2014)), and 346×260 in established models, though some manufacturers like iniVation provide resolutions that reach 640×480. Event data is reportedly often noisy (Mitrokhin et al. (2018), Milde et al. (2017)) and may require additional filtering. The principal challenge with ECs is perhaps the necessity to develop new algorithms and methods to process the novel asynchronous event stream data and successfully solve visual processing tasks as well as conventional methods do. While this remains a topic of ongoing research, various adaptations of common algorithms for pose estimation, image recognition, object tracking, etc. have been developed in addition to neuromorphic algorithms, such as those employed in spiking neural networks (SNNs).

At present, various EC models are commercially available; we mention a notable few here. Arguably the first practical EC, the DVS, implements pixels that respond logarithmically to light intensities and emit events at relative changes, reportedly achieving "orders of magnitude" lower data output rates compared to normal camera sensors (Lichtsteiner et al. (2008)). The ATIS (Asynchronous Time-Based Image Sensor) differs in its hardware design but implements the same concept, and is characterized by a distinctly higher dynamic range (143 dB vs. 120 dB) and a higher resolution (304×240 vs. 128×128), but a larger pixel area (Posch et al. (2010)). The widely-used DAVIS (Dynamic and Active-pixel Vision Sensor) succeeded the DVS, incorporating conventional frame-based output in addition to the event output, and a higher resolution (Brandli et al. (2014)). In one of our experiments, we use a DAVIS346; a model with a 346×260 resolution. The most prominent manufacturers of the aforementioned ECs are iniVation and Prophesee, though others including Samsung and CelePixel have contributed their own models.

An event can be represented as a tuple of pixel position, emission timestamp, and polarity: $e_k = (\mathbf{x}_k, t_k, p_k)$. Pixels in an EC are independently monitored to compute a measure of the difference in successive intensities, the simplest example of which is the difference in raw intensities:

$$\Delta L(\mathbf{x}_k, t_k) = L(\mathbf{x}_k, t_k) - L(\mathbf{x}_k, t_{k-1}) \tag{3.1}$$

When this difference crosses a pre-defined threshold, θ, event e_k is emitted with a positive or negative polarity, p_k, and is thus designated an ON or OFF event, respectively:

$$p_k = \begin{cases} +1, & \text{if } \Delta L(\mathbf{x}_k, t_k) > \theta \\ -1, & \text{if } \Delta L(\mathbf{x}_k, t_k) < -\theta \end{cases} \tag{3.2}$$

Facilitated by the AER and similar representations, an EC transmits a stream or vector of such e_k tuples.

It follows that event data can be emulated from conventional camera data, a method often used due to the scarce availability of or relative expense of acquiring ECs (García et al. (2016), Hu et al. (2021), Rebecq et al. (2018)). In its simplest manifestation, this method involves subtracting RGB or grayscale intensities in consecutive video frames to estimate $\Delta L(\mathbf{x}_k, t_k)$, then deriving events using the above equations. This concept is employed in many works, including this thesis project, to expedite the development and evaluation of event-based algorithms. Although it forgoes the high temporal resolution advantages of ECs, since information sampling rate is upper-bounded by the source camera's frame-rate, it facilitates preliminary applications of event-based processing prior to deployments with a real EC.

3.1.1 Event Data Representations

While ECs usually output events in a stream of e_k tuples, a variety of event data representations have been proposed for their practicality and suitability for addressing certain tasks. Some of these representations have been reviewed in G. Chen et al. (2020) and Gallego et al. (2022).

Event spike tensors (ESTs) represent counts of events within uniform time intervals and pixel locations by discretizing the temporal dimension in bins and producing $T \times H \times W$[2], disregarding event polarity (Dubeau et al. (2020)) tensors. In some publications, this is referred to as a voxel grid representation. Often, the quantities in the voxels are additionally normalized for desirable properties. This representation is convenient for conventional convolutional algorithms, such as CNNs.

Time surfaces (TS) are a 2D map representation where every pixel location is assigned the timestamp of its last event (Maro et al. (2020), Y. Zhou et al. (2021)). TSs are particularly used to encode motion histories, where the intensity of each pixel is larger the more recent the last event, i.e. the motion, in that location is. An exponential decay kernel is often applied to the time values to affect pixel intensity according to event recency.

Several representations map events to image-like arrays, which can be achieved in myriad ways. The most common involve an aggregate measure of events at each pixel, such as counts or averages of event timestamps (Mitrokhin et al. (2018)). This idea is easy to extend to similar measures, such as mean event emission rate (for e.g., by averaging event counts), but which often thus discard temporal information. Conventional brightness images have also been reconstructed from event data , such as in Scheerlinck et al. (2020), in which an RNN is trained to perform the reconstruction. Simpler representations could involve marking pixels with event polarities at every event, an idea we use in our own event representation.

We refer to Gallego et al. (2022) for an excellent and recent survey on event-based vision which delves in more detail into the topics we briefly discussed in this section.

3.2 Spiking Neural Networks

The so-called third generation of neural networks, spiking neural networks (SNN), are neuromorphic and more biologically-plausible models in which neurons communicate via asynchronous "spikes", as

[2] T is the time period, while H and W represent the height and width of the pixel array, respectively

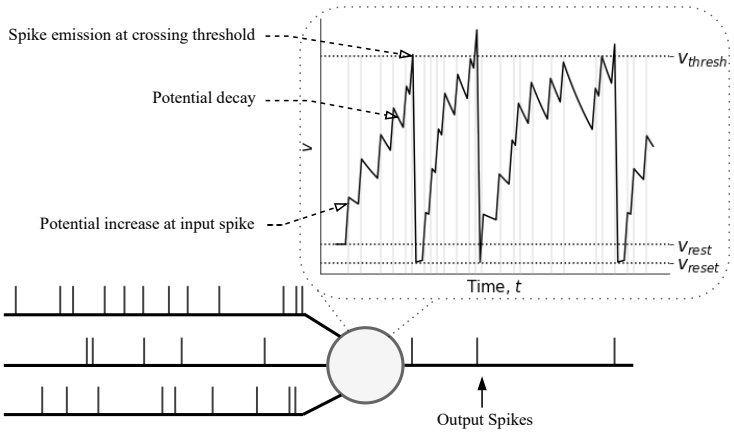


Figure 3.1: An illustration of spiking neuron dynamics.

biological neurons seem to do. Also called event-driven NNs, these models present a paradigm shift in neural processing, abandoning the synchronous propagation of real-valued signals that is characteristic of artificial neural networks (ANNs) and attempting to replicate the complex neuronal dynamics observed in the brain. Similar to ECs, SNNs have been conceptualized as a potential means towards realizing the impressive capabilities and efficiency of biological intelligence, but also present practical advantages in modern computing systems.

Figure 3.1 illustrates the dynamics of a spiking neuron. Neurons asynchronously propagate sequences of sparse, binary signals, termed spike trains, across their synapses. Each pre-synaptic spike contributes to the post-synaptic potential (PSP) or *membrane potential*, v, of a post-synaptic neuron: an internal analog representation of neuronal activation that decays over time, and an approximation of ionic concentrations across a nerve cell's membrane. A neuron whose potential exceeds a threshold, v_{thresh}, emits a spike, also called an *action potential*, and enters a short refractory period in which it is inhibited from spiking for a short duration[3]. Following a spike, the potential is reset to a baseline value, v_{reset}. Individual neurons therefore fire or are "active" only in response to a significant aggregate of recent inputs. Information can be contained in average spiking rates and/or the relative timings of spikes, a concept drawn from neuroscientific evidence (Fairhall et al. (2001)). The time-varying potential signal inherent to every neuron and the independent spiking latencies create an additional temporal dimension in spiking networks.

The now ubiquitous, second-generation NN model was devised with some inspiration from biology, particularly in structural concepts, but nevertheless fundamentally differs in computational aspects. Here, neurons communicate via real-valued, continuous signals computed by applying a non-linear (for e.g., sigmoidal) activation function to a summation of real-valued inputs. These signals can be thought of as an approximation of the firing rate of a spiking neuron. Though algorithmically expedient, this model necessitates that all neurons are constantly "active" and perform these operations with no regard to the significance of the inputs, which may be a contributing factor to the substantial power consumption of

[3]The refractory period is not depicted on Figure 3.1.

modern DNNs applied in complex problems that the brain can achieve with relative ease and efficiency. In addition, this ignores the temporal dimension in relative spike timings inherent in brain signals and which is thought to encode useful information and possibly be a critical factor in the brain's capabilities. While synaptic adjustment is known to drive learning in biological networks, the ANN learning approach we rely on the most today, the popular back-propagation rule (Rumelhart et al. (1986)) is biologically implausible. Backprop relies on real-valued neuronal signals, global error computations, and the flow of information backwards in space and time; all of which are unlikely to occur in the brain. This observation has been made by pioneers and prolific researchers of backprop (Ananthaswamy (2021), LeVine (2017)), who have suggested more biologically-plausible principles as a means to advancing machine learning. Despite the significant success of DNNs in complex, but specialized, tasks, targetting these discrepancies holds potential for enhancing computational capabilities and addressing the growing computational, power, and data requirements often imposed by these networks.

The neuromorphic computational principles of spiking networks may provide a solution for the deficiencies of modern ANNs. Through a sparse representation of information and event-driven processing, SNNs could be more efficient by processing only salient or important data, and thus lowering energy expenditure and perhaps increasing data-efficiency. Aside from potential immediate benefits, SNNs can facilitate exploring a novel computational paradigm, which may ultimately prove advantageous, if not groundbreaking, for machine learning. When implemented on embodied robotic agents, SNNs may also represent useful testbeds for neuroscientific studies (such as in the field of neurorobotics).

The novelty of this computational paradigm, however, presents the primary challenge associated with successful, practical applications of SNNs. As in the case of ECs, we are faced with the problem of devising algorithms that appropriately handle and usefully apply spiking data to AI problems. Moreover, a proper implementation of SNN dynamics and response asynchrony necessitates developing specialized neuromorphic hardware on which they can be run. These neuro-processors must abandon the familiar digital, von Neumann computing architectures (see section 2.3), which further exacerbates software development. Another challenge pertains to developing SNN learning algorithms. SNN spike trains resemble a sequence of Dirac delta functions: a non-differentiable signal. This limitation, apart from the aforementioned biological implausibility of gradient descent in the brain, motivates research into local learning rules and other alternatives. Candidates include local spike-timing-dependent plasticity (STDP), variations of Hebbian learning, and approximations of backprop.

Ultimately, as asserted in various studies reviewed in section 2.2, an immediate challenge is the achievement of similar and competitive performance to DNNs in AI problems, though there is significant ongoing research on exploring SNN formulations and learning algorithms (Pfeiffer & Pfeil (2018), Tavanaei et al. (2019)) and various propositions for spiking models and neuronal coding strategies.

3.2.1 Spiking Neuron Models

Modeling the complex dynamics occurring at neuronal synapses to a high level of detail is a difficult endeavour. Consequently, various mathematical approximations and convenient abstractions have been employed in SNN research to produce a variety of neuron models at several levels of fidelity, of which we

discuss a prominent few.

The Hodgkin-and-Huxley Model Hodgkin and Huxley presented what is often cited as the earliest formal model of a spiking neuron in the 1950s, which aimed to accurately model ionic flows across neuronal membranes in terms of differential equations describing a capacitive circuit (Hodgkin & Huxley (1952)). Through experiments on the squid giant axon, they found that the neuronal membrane contains voltage-gated ion channels that modulate the flow of sodium, potassium, and a group of less significant ions across the membrane, giving rise to the electrical and spiking activity observed in nerve cells. An electrical circuit approximating the H-H model is illustrated in Figure 3.2. In essence, the membrane is modeled by a capacitor in parallel with separate resistors, R_i, and batteries, E_i, that define the sodium (Na), potassium (K), and so-called leakage (L) currents: I_i[4]. The batteries represent the "equilibrium potential" of each ion; the difference between these potentials and the membrane potential, v, is determined by the ratio of concentrations of that ion inside and outside of the cell. The membrane potential, v, can be expressed in terms of the sum of ionic currents and any applied current, I:

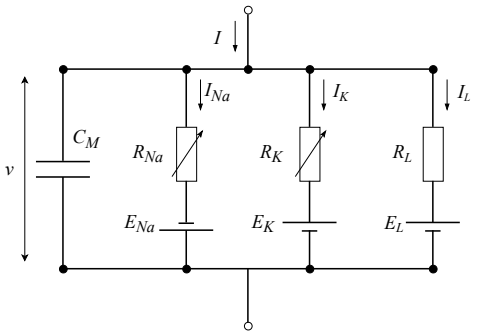


Figure 3.2: A schematic diagram of an electrical circuit that describes the Hodgkin-Huxley model. Adapted from Hodgkin & Huxley (1952).

$$C_m \frac{dv}{dt} = I - \overbrace{g_{Na}(v - E_{Na})}^{I_{Na}} - \overbrace{g_K(v - E_K)}^{I_K} - \overbrace{g_L(v - E_L)}^{I_L} \tag{3.3}$$

$$g_{Na}(v,t) = \bar{g}_{Na} m(v,t)^3 h(v,t) \tag{3.4}$$

$$g_K(v,t) = \bar{g}_K n(v,t)^4 \tag{3.5}$$

Here, $\bar{g}$ represents conductance per unit area. The authors modeled the conductance across each ionic channel in terms of gates which must all be open for current to flow. Na channels contain three "activation gates" and one "inactivation gate", while K channels contain four "activation gates", whose probabilities of being open at any time are governed by the voltage-dependent m, h, and n functions, respectively. Distinctive forms and time constants for these functions were determined from empirical data and shown to create the specific flow of ions that reliably produces the action potentials observed in neurons. The H-H model is often defined by Equation 3.3 in addition to the three differential equations describing m, h, and n. The reader is referred to the original publication for more details.

[4] where $i \in \{Na, K, L\}$

The Leaky-Integrate-and-Fire Model Other models seek to mimic these neuronal dynamics but avoid the computational complexity associated with the H-H model, particularly for applications in large networks. One such simplification is the leaky integrate-and-fire (LIF) model: the most commonly used spiking model (Rajendran et al. (2019), C. Lee et al. (2020), Dupeyroux et al. (2021)). A LIF neuron represents a leaky integrator modeled as a minimal RC circuit with an equation describing membrane voltage that can be simplified to:

$$\tau_v \frac{dv}{dt} = -(v - v_{rest}) + I(t) \tag{3.6}$$

where v_{rest} is a resting potential, τ_v is a potential decay constant, and $I(t)$ is the sum of input currents. $I(t)$ is formulated as the sum of input spikes, $S(t)$, arriving from pre-synaptic neurons indexed by i, multiplied by synaptic weights, w:

$$I(t) = \sum_{i=1}^{n_l} w_i S_i(t) \tag{3.7}$$

where $S_i(t)$ is simply 1 if a spike occurs at time t, and 0 otherwise[5]. If v exceeds v_{thresh}, a spike is emitted and v is reset to v_{reset}:

$$v(t) \leftarrow v_{reset}, \text{ if } v(t) > v_{thresh} \tag{3.8}$$

In addition, the neuron is prevented from firing again for a refractory period, T_{refrac}. The LIF neuron thus maintains a decaying memory of past inputs and implements fundamental spiking and refractoriness properties. The model is popular for its computational simplicity, although it ignores a number of neuronal characteristics (Izhikevich (2004)). In our approach, we use a version of the LIF model presented in Diehl & Cook (2015).

The Spike Response Model Gerstner's spike response model (SRM) (Gerstner (1995)) is similar in representing v as a weighted sum of input spikes, but models leakiness and refractoriness by convolving input signals with response kernels. The evolution of v can be described with:

$$v(t) = \sum_{i=1}^{n_l} w_i(\epsilon * S_i)(t) + (\nu * S_o)(t) \tag{3.9}$$

where ϵ and ν represent spike response and refractory kernels, respectively, and S_o is the neuron's output spike train. Here, synaptic weights are multiplied by the convolution of the spike signal and ϵ, which is often formulated to model an exponentially decaying contribution from each spike, but can incorporate other effects, such as axonal delays (Shrestha & Orchard (2018)). As in the LIF model, a spike is generated whenever a threshold is exceeded, whereas the refractory response following an output spike is facilitated by ν. The SRM model essentially substitutes differential equations for convolutional filters.

The Izhikevich Model The Izhikevich model (Izhikevich (2003)) was designed to offer a compromise between the biological realism of the H-H model and the computational simplicity of the LIF model. It

[5]This could be represented by a Kronecker delta function, for example.

models neuronal dynamics through the differential equations:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{3.10}$$

$$\frac{du}{dt} = a(bv - u) \tag{3.11}$$

The membrane potential depends on synaptic input currents as well as a "membrane recovery variable", u, which models the behaviour of Na and K ionic channel gates. The function and constants were selected to enable modeling the firing patterns of several known cortical neuron types, particularly through parameters a, b, and c. These include regular spiking, bursting, and "chattering". Spikes are emitted when v_{thresh} is crossed, triggering a reset of both v and u:

$$\left.\begin{aligned} v(t) &\leftarrow v_{reset} \\ u(t) &\leftarrow u(t) + c \end{aligned}\right\} \text{ if } v(t) > v_{thresh} \tag{3.12}$$

Probabilistic Models A class of probabilistic spiking models have also been proposed as alternatives to deterministic response characteristics, perhaps inherited from ANNs, particularly for properties that are favourable to learning algorithms (Jang et al. (2019)). For example, by representing spike signals through joint distributions of binary random processes, it is possible to derive surrogate gradients from the distribution functions to enable common gradient descent. Instead of deterministically, neurons can be set to spike with a probability proportional to the membrane potential, thus also mimicking noisy spiking behaviour observed in the brain. Nevertheless, probabilistic SNNs remain relatively less prevalent.

3.2.2 Spiking Data Coding Schemes

The information embedded in spike trains can be interpreted through different neural coding schemes. The concept of neural codes is inherited from neuroscientific research, in which two common problems are the encoding of stimulus signals into the corresponding neuronal response and the reverse of decoding a recorded response into the signal that gave rise to it. Decoding neuronal responses is of particular interest for recovering a useful output from an SNN, such as in our present work.

The most common coding scheme is *rate coding*, in which a neuron's response or level of activation is measured by its mean firing rate over a given duration. This can include variants such as spike counts and different methods of aggregating spike sequences. Rate codes eliminate specific spike timings and provide a scalar value that is analogous to an ANN's neuronal output. Despite essentially approximating conventional ANNs under a rate coding, SNNs can still provide a better performance vs. energy consumption and latency trade-off (Jang et al. (2019)).

With *temporal coding* schemes, temporal information is preserved by considering the absolute or relative timings of spikes in some manner. For example, the time point at which an output neuron spikes following the presentation of a stimulus can be associated with its activation level, thus potentially requiring as few as a single spike from each neuron to determine its response (Mostafa (2017)). In the

present work, we use the so-called time-to-first-spike (TTFS) or first-spike-time (FST) coding, in which the time at which a neuron first spikes represents the magnitude of its activation. Other approaches deal with inter-spike intervals, assessing neuronal responses based on the durations between consecutive spikes. Empirical observations indicate that brain computations are more likely to rely on a temporal code rather than average firing rates due to information transmission speeds; the time required to perform visual cortical computations seems to be significantly less than the time that would be required to accumulate spikes to compute a reliable average (Maass (1997)).

Rank-order coding is a form of temporal coding in which information is contained in the order in which output neurons spike, and is thus similar to FTS coding. However, most implementations of rank-order coding allow every neuron to spike only once, resulting in an informative but efficient code (Kheradpisheh et al. (2018), Kheradpisheh & Masquelier (2020)). This efficiency is due to signal sparsity and can positively impact energy consumption, since energy is expended at spike emissions.

Another common scheme is *population coding*, in which the collective responses of a group, or population, of neurons encodes relevant information. Evidently, the idea of aggregating responses is not orthogonal to the concepts of rate and temporal coding, and is rather often complementary to these coding schemes. Population coding was similarly devised from neuroscientific observations, and has been frequently applied in practical applications (Tang et al. (2020), Milde et al. (2017), Stagsted et al. (2020)).

In section 2.2, we refer to publications that review research into SNNs and provide a more comprehensive overview of the field.

3.3 Dynamic Motion Primitives (DMP)

Dynamic motion primitives enable capturing and reproducing discrete or rhythmic motions using a set of differential equations that produce stable global attractor dynamics (Ijspeert et al. (2013)). For discrete motions, DMPs model the evolution of a position variable, $\mathbf{y}$, from its starting position to a goal position, $\mathbf{g}$, over time through equations describing a linear spring-damper system augmented with an additional non-linear term:

$$\tau\ddot{\mathbf{y}} = \alpha_{\mathbf{y}}(\beta_{\mathbf{y}}(\mathbf{g} - \mathbf{y}) - \dot{\mathbf{y}}) + \mathsf{f}(\mathsf{s}) \tag{3.13}$$

$$\tau\dot{\mathsf{s}} = -\alpha_{\mathsf{s}}\mathsf{s} \tag{3.14}$$

where $\alpha_{\mathbf{y}}$ and $\beta_{\mathbf{y}}$ control spring and damping characteristics and τ is a time scaling constant. The first equation, termed the "transformation system", describes the accelerations (and, by extension, velocities and positions) that drive the system from the initial to the goal position. $\mathsf{f}(\mathsf{s})$ represents a non-linear "forcing term", which describes the shape of the trajectory through a superposition of basis functions and depends on a phase variable, s:

$$\mathsf{f}(\mathsf{s}) = \frac{\sum_i \mathsf{w}_i\psi_i(\mathsf{s})\mathsf{s}}{\sum_i \psi_i(\mathsf{s})} \tag{3.15}$$

where ψ_i are basis functions (usually of a Gaussian kernel) and w_i are learnable weights that capture the shape of the desired trajectory. A demonstrated trajectory can be learned by sampling positions, velocities, and accelerations then solving for a least-squares solution w_i using linear optimization. The

second equation models the so-called "canonical system", which describes the evolution of phase variable `s` from 1 to 0 (the start to the end of the motion). The function `f` depends on `s` instead of a time variable in stepping through the sequence of functions that reproduce the trajectory shape, which enables scaling the trajectory in time by modifying τ.

4

Proposed Solution

This chapter discusses the conceptual and technical details of the neuromorphic approach we propose as a solution to problem of obstacle avoidance on a robot manipulator. In section 4.1, we present the pipeline constituting our SNN-based obstacle avoidance module, describing each component and the computations performed at each stage from the visual inputs to the trajectory adaptations that enable real-time avoidance. After developing the theoretical foundations of these components, we describe their software implementations in section 4.2, providing details on their integration in a unified framework, notable functionalities, and parameters. The implementation details presented in this section may not be necessary for a high-level understanding of our pipeline. The chapter closes with concluding remarks.

4.1 Proposed Approach

In this section, we present our approach to the problem of incorporating event-based vision and spiking neural networks (SNNs) in adaptive motion execution for obstacle avoidance. We rely on dynamic motion primitives (DMPs) to generate trajectory plans for moving a manipulator between arbitrary initial and goal positions in accomplishing tasks such as reaching for an object. The DMP formulation supports additive acceleration terms that modify position variable evolution and thus trajectory plans online, which we utilize to inject obstacle avoidance information to adapt pre-planned trajectories and guide the arm's motion away from obstacles. This obstacle avoidance information is obtained by continuously processing visual, event-based data within an SNN, then decoding output neural activation maps into obstacle avoidance accelerations. The decoding procedure involves a potential field (PF) representation for computing the most favourable obstacle avoidance direction. Therefore, real-time, online trajectory adaptation is achieved by utilizing events induced by the relative motion of objects and the spatio-temporal filtering properties of SNNs to extract reactive motions that modify high-level motion plans to react to obstacles while maintaining progress towards the intended goal.

A core aspect of the proposed approach is the synergy between global planning and local corrections for goal-directed, obstacle avoidance motions: an idea that has been advocated in the literature. In the Handbook of Robotics, Minguez et. al. remark about the importance of combining "the global knowledge given by motion planning and the reactivity of the obstacle avoidance methods" for motion system design (Minguez et al. (2016)). The alternative of a planner completely re-planning when obstacles are encountered could impose higher computational expense and latencies (D'Silva & Miikkulainen (2009),

Feng et al. (2020))), which are particularly undesirable in dynamic environments. Instead, we utilize the DMP as an adaptive planner, where perceived obstacles are handled by appropriately adjusting the next waypoint during execution, and the DMP's global attractor dynamics ensure that the trajectory gracefully returns to the original path.

We use spiking neural networks (SNNs) to process event data. This is partially motivated by the desire to study the more biologically-inspired form of neural networks, but also for its suitability for event-based vision. Classical computer vision algorithms, including CNNs, are not directly applicable to event data (G. Chen et al. (2020), Vitale et al. (2021)). On the other hand, the naturally compatible SNNs are designed to process discrete, asynchronous, signals (Taunyazov et al. (2020), Salvatore et al. (2020)), and may be key in achieving compelling real-world applications of ECs (Davies et al. (2021)). The combination of events and spiking neurons holds the potential for capturing temporal information relating to obstacle avoidance, such as through the decaying influence (neural activation) of an obstacle that has just been observed. In Salvatore et al. (2020), the properties of SNNs allowing for a form of "temporal attention" are studied and contrasted to traditional LSTMs, which are considerably more computationally expensive. The analog SNN dynamics generally induce temporal filtering properties that may negate effects of noise and/or insignificant motions in event data (refer to the discussion on SNN spatio-temporal filtering in section 4.1.3).

Next, an overview of the proposed approach and its main constituents is presented, followed by more detailed discussions of each component: event camera emulation, convolutional spiking neural networks, the obstacle avoidance components, and trajectory planning and motion control.

4.1.1 Overview

The proposed approach can be represented as a modular pipeline of specialized components. Figure 4.1 depicts the pipeline in a block diagram. The top block illustrates a high-level view of the proposed approach: a module which is supplied with a pre-planned motion trajectory and produces an adapted trajectory in a closed-loop, online procedure. The bottom part of the figure shows an expanded view of the module, whose main sub-components can be divided into:

1. **EC/EC Emulator:** An event camera, or a camera from which event data is emulated, which produces event data in an "event image" representation.
2. **Convolutional SNN:** An SNN whose neurons are arranged in convolutional layers, and which processes incoming "event images" over time.
3. **Obstacle Avoidance Component:** A component that decodes SNN outputs (spike trains) into obstacle avoidance velocities/acceleration, using a PF method.
4. **Motion Controller (DMP):** The motion controller that generates the end-effector positions (and velocities) following a planned trajectory, and adjusts the plan according to the output of the obstacle avoidance component.

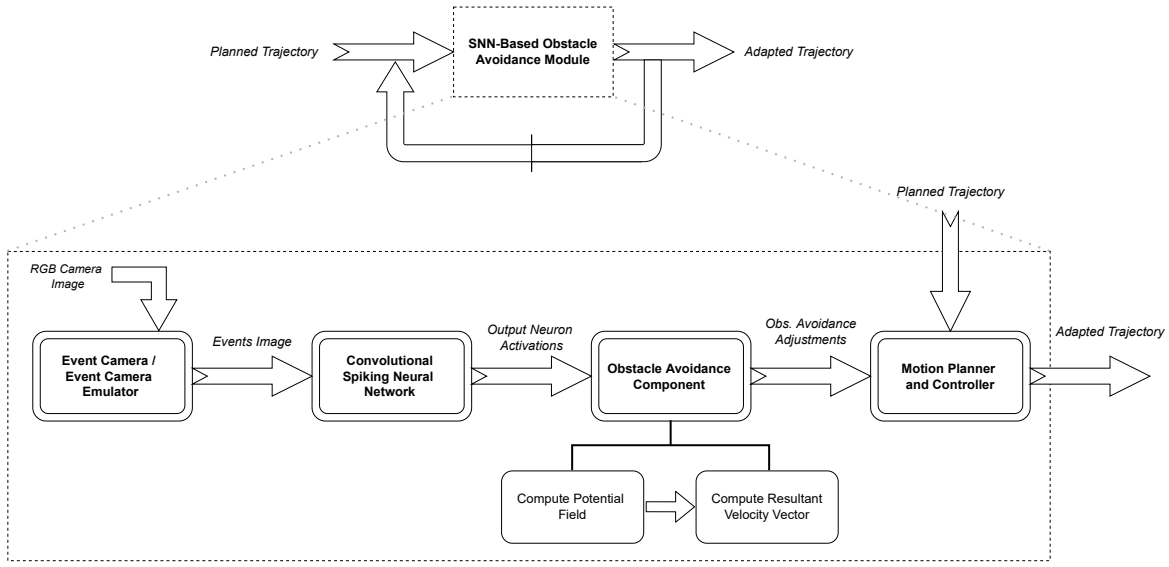


Figure 4.1: A block diagram depicting the proposed SNN-based obstacle avoidance module (top) and a break-down of its main components (bottom)

4.1.2 Event Camera Emulation

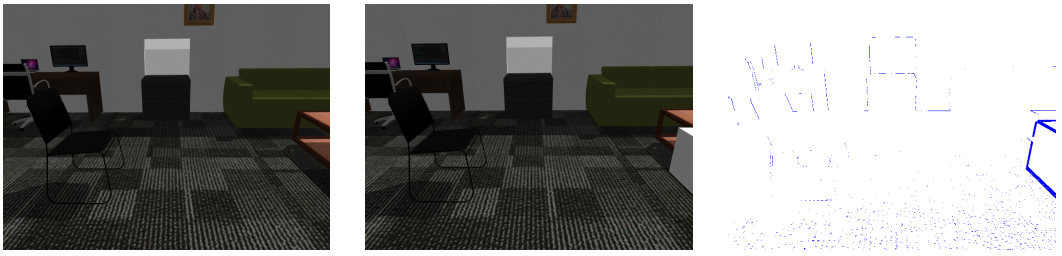
The primary sensory input to the obstacle avoidance module is event data. As outlined in section 2.1, ECs possess attributes that are potentially beneficial to autonomous systems in visual processing tasks, including higher energy efficiency, low latencies, high dynamic ranges, and robustness to motion blur. These attributes arise from the unique paradigm of pixels transmitting data independently and asynchronously in response to exclusively significant inputs as inspired by biological retinas. Research on ECs is still in relatively early stages and mostly involves aerial robots. This motivates the application and study of event data in a manipulation scenario in this thesis.

In the present work, event data is derived from conventional RGB camera data through an emulation software component: a method often used in the literature (see end of section 2.1). Following the fundamental operating principles of an event camera, events can be generated from the thresholded difference in intensity values at every pixel between consecutive timesteps. The resulting data resembles the output of a real EC, such as a DVS. Section 2.1 contains a review of existing event emulation implementations and their perceived deficits (coupling to rendering engines, no ROS support, etc.), which motivate developing the *event_camera_emulation* component presented in this section.

Referring to section 3.1, an event $e_k = (\mathbf{x}_k, t_k, p_k)$ is emitted at pixel position $\mathbf{x}_k$ at time t_k with polarity p_k if the difference in intensities $\Delta L(\mathbf{x}_k, t_k)$[1] exceeds a threshold, θ.

Here, we utilize an event image representation, where an image-like frame composed of events is created

[1] An arbitrary difference measure: $d(L(\mathbf{x}_k, t_k), L(\mathbf{x}_k, t_{k-1}))$.



(a) RGB Image 1 (b) RGB Image 2 (c) Event Image

Figure 4.2: Emulated event image example: motion of an object on the right in consecutive RGB images. Here, any event (ON or OFF) is indicated by a blue pixel on the "event image". This was captured as the camera moved forward, thus inducing some events on the edges of distant objects.

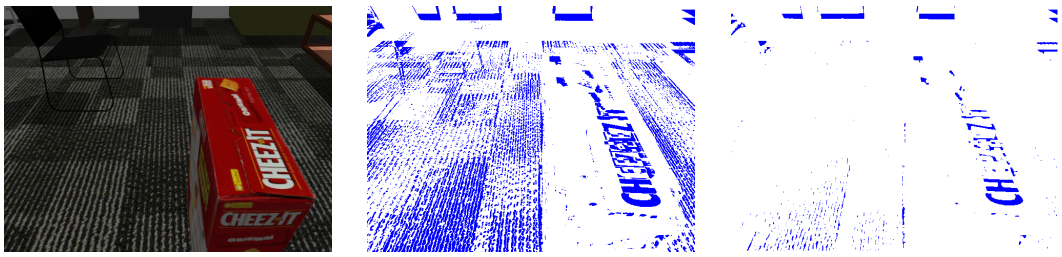
by placing p_k of every event at the respective pixel location. The resulting "event image", I_e, matches the shape of the source RGB images, but contains one-dimensional values, $i_k \in \{+1, 0, -1\}$. Figure 4.2 visualizes an event image derived from two consecutive RGB images. For most purposes, the distinction of event polarity is of little importance and OFF events are either ignored or treated the same as ON events, as we do here (and similar to Dubeau et al. (2020) and Maro et al. (2020)). This representation is adequate for processing in the next component of the pipeline: convolutional SNNs.

Limitations of Emulation

It is worth mentioning that the emulation of events from conventional camera data presents a limitation. Deriving events from differences in consecutive RGB frames places an upper bound on the rate of event generation: the camera frame rate. This diminishes the advantages of asynchrony of pixel events and the theoretically higher transmission rates in comparison to conventional camera pixels. In addition, presumably superior dynamic ranges and robustness to lighting conditions would not be observed. Nevertheless, this method provides a reasonable approximation for demonstrating and presenting elementary arguments for the approach proposed in this thesis. We verify the validity of this approximation by comparing the output and consequent task performance of our emulator to those of a real EC in section 6.7.

Filtering Event Noise: Binary Erosion

In some cases, it is useful to apply filters on event images to eliminate undesirable or noisy events, such as when the environment contains distracting background textures. We have found that a binary erosion filter helps in producing cleaner event images in cases where textures or surfaces in the environment induce too many events that may lead to over-reactive responses. Figure 4.3 depicts such a case, in which the background (4.3a) produces a high concentration of events for small relative motions (4.3b), especially from the rough carpet. However, applying a binary erosion filter to the image reduces the occurrences of insignificant background events (4.3c). The filter consists of a fixed-size kernel of 1's that scans the image and eliminates points, i.e. events ($abs(1) \rightarrow 0$), if the surrounding region does not match all of the kernel's values. This effectively removes events for which the local region does not contain sufficiently many



(a) RGB Image (b) Raw Event Image (c) Event Image, with binary erosion

Figure 4.3: An example of a binary erosion filter applied to an event image. (a) shows a scene containing rough textures and visual distractors that lead to high event emission rates for background textures (b). Applying binary erosion reduces the effects of irrelevant background events.

other events, achieving a noise filtering effect[2]. The idea of filtering events, particularly for suppressing background artifacts, has also been explored in Maro et al. (2020).

4.1.3 Convolutional Spiking Neural Networks

An SNN propagates spike trains instead of constant real-valued signals and is characterized by internal analog dynamics. SNN spikes resemble EC events and suggest an intuitive relationship between event-based vision and SNNs: the latter are naturally suited to process asynchronous binary signals. Indeed, SNNs have been developed to mimic the information processing mechanisms observed in biological neural networks, such as those traveling from photoreceptor cells to the visual cortex and within it. It is therefore worth considering a neuromorphic neural network for processing neuromorphic sensor data when studying possible directions toward mammal-like visual processing capabilities.

Input Data Encoding

We utilize a Poisson process spike generation model to induce spikes in the SNN input layer from incoming event images. External inputs to SNNs are often encoded into spikes using one of various temporal coding methods, such as single-spike or rank-order codes, or rate coding methods, such as random processes. A Poisson process presents a useful stochastic approximation of biological neuron firing activity, in which the generation of each spike is assumed to depend on some firing rate, r, and be independent of all other spikes (Heeger et al. (2000)). In particular, the number of spikes in a given time interval, δt, is an independent Poisson-distributed random variable. A consequent useful property of the Poisson process is that the length of inter-spike intervals (ISIs) can be drawn from independent exponential random variables with means $1/r$. Therefore, for a given time period, T, a Poisson process can be used to compute

[2]Note that increasing the intensity difference threshold, θ, can reduce the instances of sporadic event emissions. In practice, however, we have found that very high thresholds could eliminate useful event information, since all potential events are equally suppressed. The binary erosion filter, on the other hand, exploits the local information around events, achieving a more reliable filter in this case.

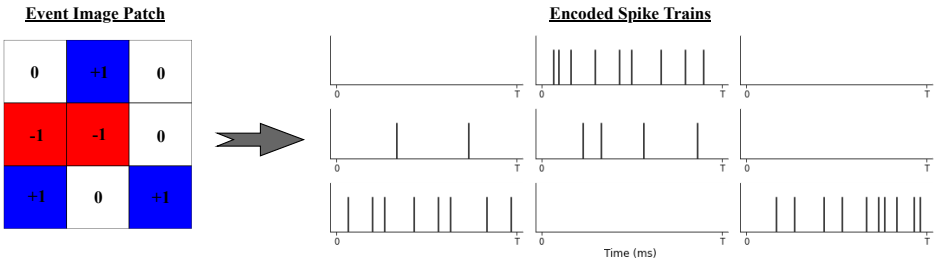


Figure 4.4: An illustration of the Poisson spike trains generated from events at 9 input neurons. Positive and negative events are shown in blue and red, respectively. Note the lower spiking frequency at negative events.

inter-spike intervals that define a Poisson-distributed sequence of spikes for a given rate, r (also referred to as intensity) determined by the input.

Given event image I_e of size (w, h), we assign rate values according to:

$$I'_e(\mathbf{x}) = \begin{cases} r_{ON}, & \text{if } I_e(\mathbf{x}) = +1 \\ r_{OFF}, & \text{if } I_e(\mathbf{x}) = -1 \\ 0, & otherwise \end{cases} \tag{4.1}$$

where r_{ON} and r_{OFF} are tunable parameters representing firing rates in Hz. In our experiments, these parameters are set such that OFF events induce approximately half the stimulation that ON events do. Subsequently, the spike train entering each input neuron across T is drawn from a Poisson process, which determines the presence (1) or absence (0) of a spike within each discrete time interval δt. These sequences of spikes follow the average firing rate but have random spike timings. Figure 4.4 illustrates an example of a 3×3 event image patch and the spike trains generated at every pixel/input neuron location. The result is a binary matrix S of shape $w \times h \times T$, where each element s_t^{ij} defines whether neuron ij receives an input spike at timestep t[3].

The stochasticity of Poisson spike trains could potentially ignore the utility of precise spike timings in SNN processing. On the other hand, the relaxation of precise inter-spike intervals may provide a closer analog to the stochasticity observed in biological neural dynamics, in addition to possibly demonstrating robustness to (or even exploiting) apparent noise in propagated signals. In general, Poisson spike trains are sufficient for the purposes of inducing input spikes in the proposed approach.

Convolutional Network Topology

The encoded spike trains, S, are then inputs to the convolutional SNN, which resembles a CNN in architecture and weight sharing principles. Similar to CNNs, neurons in each layer are arranged in a two-dimensional grid and information is propagated through convolution operations. Here, the input to

[3]Note that this assumes a constant firing rate over T.

each neuron is a "pre-activation" (in conventional terms) computed by convolving the spiking output of neurons in the previous layer that occupy the target neuron's receptive field and a kernel matrix of shared weight parameters, K. This is encapsulated in the following, where the pre-activation of a neuron in layer $k+1$, and index (i, j) at a given time-step is computed by convolving a kernel with spike trains arriving from neurons in the previous layer, k:

$$act_t^{i,j,k+1} = (K * S)(i, j) = \sum_m \sum_n S(i - m, j - m)K(m, n) \tag{4.2}$$

(This formula adapts the definition of conventional convolutions provided in Goodfellow et al. (2016).) While the tunable weights can have real values, the values S are strictly binary. The pre-activation value adds to the membrane potential of the neuron. Figure 4.5 depicts the result of a convolution operation involving a 3×3 group of input neurons and a 2×2 kernel.

The network designed by cascading convolutional layers of spiking neurons operates under the same principles as CNNs and is similarly well-suited to inputs derived from images. However, the spatio-temporal distributions of spikes within the network lead to novel dynamics from which interesting properties may arise (consider, for example, the reportedly superior representational power of SNNs mentioned in section 2.2). In particular, the convolutional operator and the analog dynamics of spiking neurons create a form of spatio-temporal filter, where signals that are especially persistent in space and time are selectively propagated. Moreover, spiking neurons possess a form of memory in the decaying potential which reflects recent levels of stimulation: generally, the network is more active around areas where motion had just been observed, and the attention gradually decays with its absence over time. We use simple layers with only convolutional operations that progressively shrink the feature space, such that the output layer's dimensions (i.e. the number of output neurons) are smaller than those of the input layer. The eventual output of the network are spike trains originating from the output neurons, which are used to derive obstacle avoidance in the next component of the pipeline.

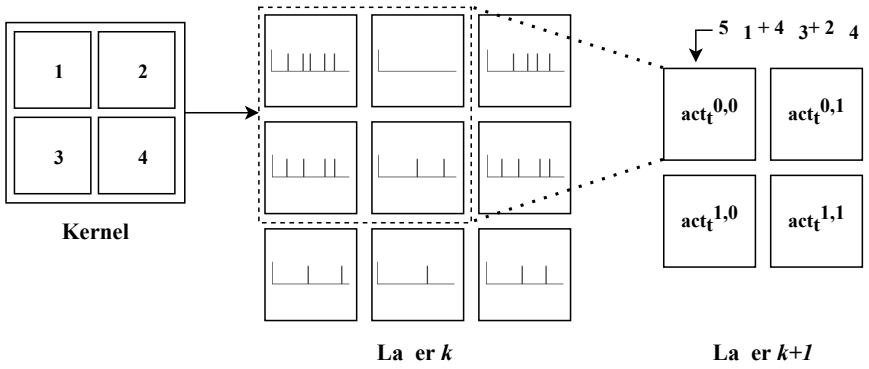


Figure 4.5: An illustration of convolving a 2×2 kernel with a 3×3 layer (k) of spiking neurons, whose output spike trains are depicted within each cell. Assuming a stride of $1x1$, the result is the pre-activations of neurons in layer $k+1$ (see Equation 4.2. The summation resulting in $act_t^{0,0,k+1}$ is shown above the cell.)

The SNN weights are set to fixed, random values. Non-plastic SNN connections have been employed in reviewed SNN research, such as in Risi et al. (2020) (see section 2.2) and Milde et al. (2017) (see section 2.4), which demonstrated the achievement of desired behaviours solely due to the properties of spiking dynamics. Similarly, we presently involve no learning capabilities, and instead investigate how robust obstacle avoidance performance is to different random, but fixed, "features" that manifest through the randomly sampled weight values. Future extensions will include incorporating weight adjustment strategies through, for example, traditional SNN learning rules such as STDP and supervised variants, or reinforcement learning. The weight values are initialized by sampling from a standard uniform distribution, scaled by the weight factor, w_c:

$$W \sim \mathcal{U}(0, w_c) \tag{4.3}$$

4.1.4 Obstacle Avoidance Component

The obstacle avoidance component decodes the output of the SNN into meaningful obstacle avoidance signals. Given output spike train, S, this stage involves extracting indications of obstacle presence or motion from spiking activity and deriving velocity/acceleration values that can be used to adapt the planned motion trajectory. To that end, we utilize a first-spike-time representation of output spiking activity and a potential fields method.

SNN Output Representation

The information contained in spike trains can be represented in different ways, including aforementioned temporal and rate coding schemes. The choice of representation is important for appropriately decoding the SNN response. We use first-spike-time (FST) temporal coding to interpret the output spiking activity (Tuckwell & Wan (2005)), which has been applied in various works (Göltz et al. (2021), Liu et al. (2021)). Within this scheme, the time until a neuron's first spike after stimulus presentation fully determines the magnitude of stimulation: the earlier a neuron first spikes, the more stimulated it is. This is intuitive, since receiving a large number of input spikes (high stimulation) is expected to raise a neuron's membrane potential faster and thus lead to an earlier firing time. When applied to the output of our SNN, the FST code provides a *neural activation map*, as illustrated in Figure 4.6. Significantly intense neural activation in a given neuron is expected to indicate the persistent presence and (relative) motion of a perceived object in regions of the input image for which that neuron is in the effective receptive field. This is a result of the spatio-temporal filtering properties of the C-SNN and provides useful indications of promising obstacle avoidance directions.

Other common codes include the absolute or average number of spikes in a given time period. Like the spike count or average, time to the first spike can indicate the neuron's level of stimulation; however, the proposed FST representation only necessitates flagging the first spike for each neuron, as opposed to waiting until all spikes in a given time window are accumulated. Therefore, within the present approach, either representation may provide the same information, but the FST code has the potential of reducing "time-to-solution" or "energy-to-solution" and could thus be more efficient (Göltz et al. (2021)). Note that

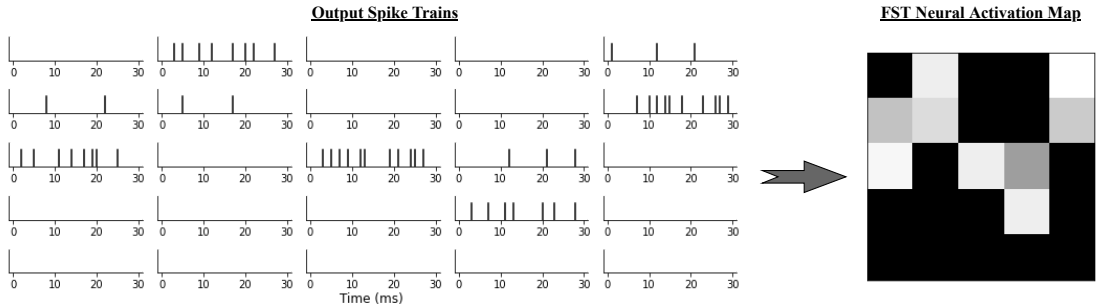


Figure 4.6: A visual depiction of the first-spike-time (FST) code applied to 25 output neurons. The brightness of each cell in the right corresponds to that neuron's FST and thus activation magnitude. Note, for example, that the top-right neuron has a higher activation due to an early first spike, although the neuron just under exhibited a higher spike count.

the FST method has been often used in classification problems in which an input stimulus is expected to yield a single inference output, where the time point at which the input stimulus was presented is clear. In contrast, our SNN runs continuously; therefore, the FSTs are calculated with respect to the time point at which the last event image was presented.

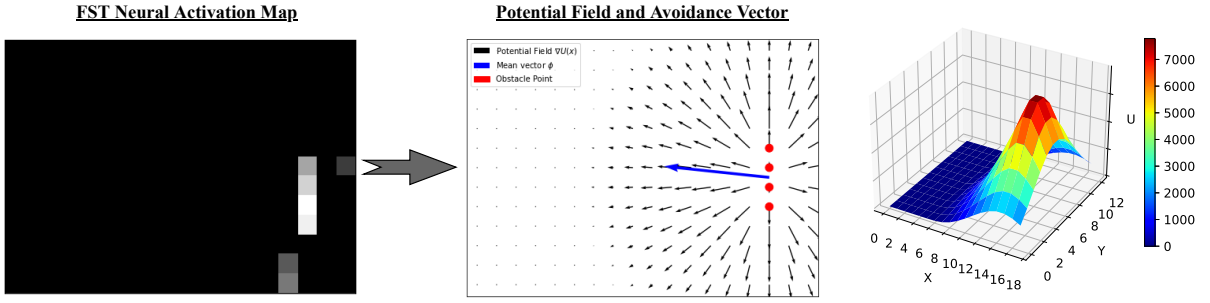
Computing Obstacle Avoidance Direction Using Potential Fields

After acquiring an output neural activation map, the next step involves computing a motion vector that is conducive to avoiding any supposed obstacles. As previously alluded to, the output activation map can be conceptualized as a downscaled version of the original input event image, filtered in the space and time dimensions to indicate approximate locations of persistent obstacles while removing potential noise. Therefore, we regard regions with high activations in the map as "obstacle points", and conceptually project their positions to the original image space. In order to derive an avoidance motion vector, we utilize a method that can represent and aggregate the obstacle points' spatial influences and compute a motion direction that maximizes movement away from these points: artificial potential fields.

Artificial potential fields (PFs) are used to create fields of attractive and repulsive "forces" overlaid on a robot's environment to drive goal reaching and obstacle avoidance behaviours, respectively (Khatib (1986)). In our approach, we construct a PF on the derived neural activation map, where the obstacle points are set to exert repulsive forces. To that end, the potential is computed using the formulation presented in Park et al. (2008). For an arbitrary point on the field, $\mathbf{x}$, the potential is determined by the distance to an "obstacle point", $p(\mathbf{x})$ according to:

$$U(\mathbf{x}) = \begin{cases} \frac{\eta}{2}\left(\frac{1}{p(\mathbf{x})} - \frac{1}{p_0}\right), & \text{if } p(\mathbf{x}) \leq p_0 \\ 0, & \text{if } p(\mathbf{x}) > p_0 \end{cases} \tag{4.4}$$

where p_0 denotes an obstacle's radius of influence and η is a constant gain. If multiple obstacle points are perceived, then the potential at point $\mathbf{x}$ is simply the aggregate of their contributions, i.e. for n obstacle



(a) Converting the neural activation map to a potential field

(b) 3D potential field

Figure 4.7: An illustration of the potential field computed from the SNN's output neural activation map. The potential field is visualized in 2D (4.7a) and 3D (4.7b), the latter showing the slope that leads to the selected velocity vector (blue).

points: $U(\mathbf{x}) = \sum_{i=1}^{n} U_i(\mathbf{x})$. The negative potential gradient, $-\nabla U(\mathbf{x})$, provides reliable estimates of directions leading away from high potential regions. Figure 4.7 shows an example of a PF applied on a neural activation map, yielding a gradient field (black arrows) which points away from the aggregated obstacle points (red). The mean negative potential gradient, $\tilde{\phi} = \overline{-\nabla U(\mathbf{x})}$, then provides an average motion vector that incorporates all potential across the field and estimates an optimal direction for avoiding the perceived obstacles. In Figure 4.7a, this is visualized as a blue arrow.

The obstacle avoidance component therefore decodes the SNN output into motion vector $\tilde{\phi}$ which provides the information needed to most adequately adapt the current motion plan to avoid the obstacles.

4.1.5 Trajectory/Motion Planning and Control

The aforementioned components of the pipeline all operate during the motion loop governed by the trajectory planning and control component. This component produces a planned trajectory prior to the robot's execution and executes the plan by controlling the robot's velocities as it moves through the designated waypoints. However, it is also capable of utilizing feedback from the obstacle avoidance component to adapt the motion plan online, by deviating from the original path to avoid obstacles, while maintaining progress towards the original goal. Therefore, it seamlessly integrates higher-level planning and lower-level, reactive motions to simultaneously accomplish both objectives. An integral aspect of this component is a dynamic motion primitive (DMP), which enables generating a trajectory plan (i.e. waypoints) given a goal position and parameters describing the desired trajectory shape, and whose formulation is conducive to online trajectory adaptation. The robot can then sequentially follow the specified positions along the trajectory through velocity commands generated by a simple PID controller.

Dynamic Motion Primitives (DMP)

Dynamic motion primitives are useful in modelling the evolution of a point's position over time in a set of differential equations that produce stable global attractor dynamics (see section 3.3). In previous work, we

utilized DMPs for reproducing learned end-effector trajectories while generalizing over starting positions and goals (Abdelrahman et al. (2020)). DMPs have several convenient characteristics that include:

- Translational invariance and generalization to arbitrary initial and goal positions
- Stable and guaranteed convergence to the goal position as $\mathsf{s} \to 0$ and $\mathsf{f}(\mathsf{s})$ vanishes
- The capacity to reproduce arbitrary trajectory shapes that can be learned by adjusting weights w_i
- The extensibility of the transformation system equations by adding task-related acceleration terms

The final property is particularly important for our proposed approach. Since the trajectory of position variable $\mathbf{y}$ can be altered by adding terms to Equation 3.13, an objective that is secondary to reaching the goal position could be accomplished by adding appropriate acceleration values during the evolution of $\mathbf{y}$. The stable attractor dynamics of the DMP guarantee eventual convergence to the goal despite perturbations, thus effectively enabling the high-level motion plan to be adjusted through deliberate perturbations while maintaining the original task target. Indeed, we utilize this property to adapt DMP-planned, goal-directed trajectories online through the output of the obstacle avoidance component.

In the present work, a DMP controls the positions of an end-effector, $\mathbf{y} = [x, y, z]^T$ as it progresses towards goal $\mathbf{g}$. The shape of the trajectory is fixed a priori i.e. w_i are given and not changed in our approach. The reason is two-fold: i) the capability of generalizing a given trajectory shape to arbitrary initial and goal positions provides sufficient adaptivity, and ii) trajectory shape adaptation is externally achieved as a consequence of incorporating the obstacle avoidance feedback. We augment equation 3.13 with an additive obstacle avoidance acceleration term, $\boldsymbol{\phi}$:

$$\tau\ddot{\mathbf{y}} = \alpha_{\mathbf{y}}(\beta_{\mathbf{y}}(\mathbf{g} - \mathbf{y}) - \dot{\mathbf{y}}) + \mathsf{f}(\mathsf{s}) + \boldsymbol{\phi} \tag{4.5}$$

The instantaneous value of $\boldsymbol{\phi}$ is directly derived from the obstacle avoidance component's output, $\tilde{\boldsymbol{\phi}}$. The latter expresses a motion vector in the image (or neural activation map) space which is transformed to the end-effector's operational space:

$$\boldsymbol{\phi} = T^{ee}_{camera}\tilde{\boldsymbol{\phi}} \tag{4.6}$$

In our case, the camera is positioned such that the image's u-axis (horizontal) and v-axis (vertical) align with the end-effector's y-axis and z-axis, respectively, yielding the simple expression:

$$\boldsymbol{\phi} = \begin{bmatrix} \phi_x \\ \phi_y \\ \phi_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \tilde{\phi_u} \\ \tilde{\phi_v} \\ 0 \end{bmatrix} \tag{4.7}$$

The $\boldsymbol{\phi}$ term contributes accelerations that lead to motion away from obstacles according to vector $\tilde{\boldsymbol{\phi}}$.

Note that $\tilde{\boldsymbol{\phi}}$ is computed from a camera image and is thus strictly two-dimensional; as a result, $\boldsymbol{\phi}$ inherits the same constraint. For a camera mounted at the end-effector, such that image plane is parallel to the end-effector's y-z plane and perpendicular to the forward-facing x-axis, the consequence of this is that obstacle avoidance vectors are constrained to the y-z plane. This is reflective of the fact that

the sensor (conventional or event camera) lacks depth perception, and thus can not be used to compute obstacle avoidance motions that are perpendicular to the image plane (through conventional means). Nevertheless, obstacle avoidance behaviour resulting from this approach is sufficient for the class of tasks targeted in this work, where the end-effector moves forwards towards a goal and the robot is expected to avoid obstacles within the FOV of a forward-facing camera.

With the integration of the decoded SNN output into the adaptive DMP, the obstacle-avoiding trajectory is then executed by following the positions integrated from Equation 4.5 with velocity commands from a PID controller.

PID Controller

The final stage of the motion loop involves following trajectory positions $\mathbf{y}$ set by the DMP, which can be accomplished using PID control. Proportional-integral-derivative (PID) control utilizes feedback to adequately control a system's response in moving from a current value to a setpoint by calculating the error and applying corrective control actions based on the weighted sum of proportional, derivative, and integral terms. Here, we design a PID controller to compute velocity commands (the control actions) that move the robot's end-effector from its current position to the next planned DMP position (the setpoint) during task execution. Given current position $\mathbf{y}(t)$ and the target position $\mathbf{y}_{target}$, the error is:

$$\mathbf{e}(t) = \mathbf{y}_{target} - \mathbf{y}(t) \tag{4.8}$$

The PID velocity control is then described by the following equation:

$$\mathbf{v}(t) = K_p\mathbf{e}(t) + K_i \int \mathbf{e}(t)dt + K_d \frac{d\mathbf{e}}{dt} \tag{4.9}$$

K_p, K_d, and K_i are constant gains which control the contributions of each term. The first term is directly proportional to the magnitude of the absolute error. The integral term accumulates a history of past error values and serves to apply corrections that eliminate residual errors. The derivative term exerts a damping effect proportional to the rate of change of the error, which provides an estimate of the future trend of the error. Figure 4.8 depicts a block diagram of the control system.

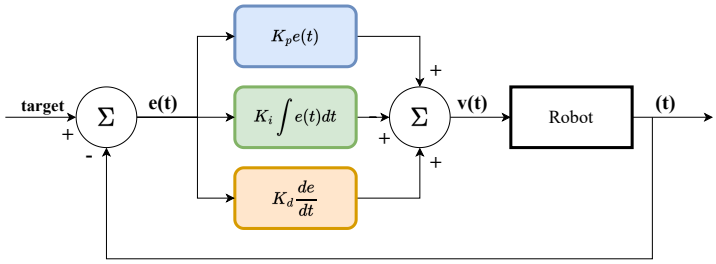


Figure 4.8: A block diagram depicting our PID controller. The proportional, integral, and derivative functions are shown in colored blocks.

By tuning respective gains, the PID controller can be optimized to improve properties such as the smoothness and stability of motions. These properties in turn influence metrics such as the predictability, naturalness, and safety of the motions, and are therefore essential parameters. In addition, the tight integration between motion and sensor-based obstacle avoidance corrections in the proposed approach requires corrective motions to be responsive and reliable, since they otherwise risk not adequately utilizing the feedback provided by the obstacle avoidance component.

Figure 1.4 provides a visual illustration of the full pipeline.

4.2 Implementation

This section summarizes the details of the implementation of the proposed approach. The components described in section 4.1 are implemented as ROS nodes and communicate information asynchronously through ROS topics. Section 4.2.1 details each ROS node's responsibilities, functionalities, and input and output data. We use the ROS Noetic Ninjemys distribution[4] and Python 3.8 for development. In order to initially test the implementation in simulation before transferring to a real robot, we set up and augment a Gazebo simulation of the Kinova Gen3 arm, the details of which are discussed in section 4.2.2.

4.2.1 ROS Components

Figure 4.9 shows a graph of the ROS nodes and the main communication channels between them (in the form of ROS topics and services) involved in running the SNN-based obstacle avoidance pipeline in simulation. When running the pipeline on a real robot, the gazebo node is substituted by other nodes, but the obstacle avoidance pipeline remains the same. In the following, we describe each of the nodes (depicted in blue), including details of their implementation, functionalities, communications, and parameters.

Event Image Streamer

Our *event_camera_emulation* python package provides functionalities for generating, streaming, and visualizing event data derived from RGB images from a camera or ROS publishers. These functionalities are implemented within the `EventCameraEmulator` class whose primary functions are depicted in Listing 1. The `get_events_image_rgb` and `get_events_image` functions produce event images (as described in section 4.1.2) from input RGB or grayscale images. These functions depend on secondary functions such as `compute_thresholded_diff_rgb_multi_channel` which define different strategies for computing events. A `get_visual_events_image` function enables visualizing event data in a format similar to the output of real ECs, where ON and OFF events are represented as blue and red pixels, respectively (the event images displayed in Figure 4.3 and elsewhere in this report have been created in this manner).

An `event_image_streamer` ROS node within the package handles capturing RGB image frames, creating event images, and publishing the event data. In our experiments, we use the `event_image_streamer` to process the latest consecutive images published by the mounted camera in "sensor_msgs/Image" messages on the ROS topic: "/camera/color/image_raw". The node employs an instance of the `EventCameraEmulator`

[4]The latest version, at the time of writing.

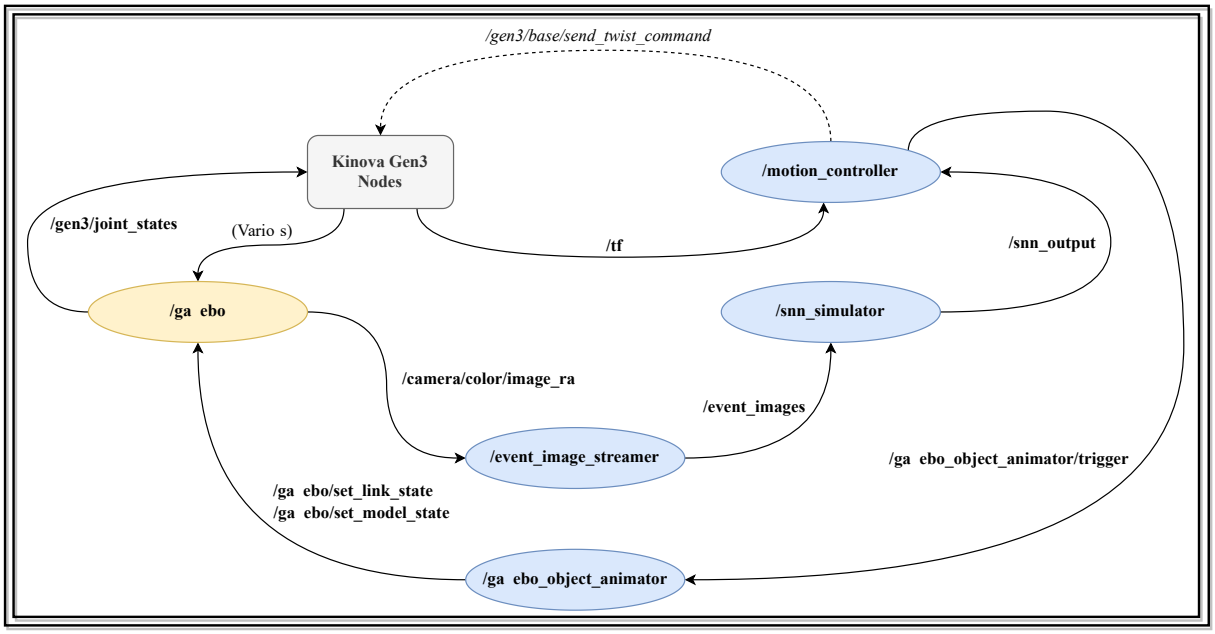


Figure 4.9: A graph of the main ROS nodes and the topics/services they communicate on. Services are depicted with dotted arrows and italics. The multiple nodes associated with the Kinova arm drivers (from the *ros_kortex* package) have been reduced to a single "Kinova Gen3 Nodes" block in the diagram.

class to produce event data based on specified parameter values. Table 4.1 lists the parameters that customize the `event_image_streamer`'s behaviour. The node then publishes event images and a user-friendly, visual representation on ROS topics "/events_images" and "/visual_events_images", respectively, in a continuous loop. Since the emulator depends on source image inputs, the publishing rate is upper-bounded by and often matches that of the source images.

Table 4.1: Parameters of the `event_image_streamer` node

Parameter	Description
theta	Event emission threshold, θ
compute_from_rgb	Whether to compute events from RGB or grayscale images
rgb_multi_channel	Whether an event is emitted when θ is exceeded in every color channel (or any one channel)
record_off_events	Whether to record OFF events
register_off_events_as_on	Whether to treat all events as ON (discarding polarity information)
use_log_diff	Whether to compute differences in log, instead of absolute, intensities
filtering_strategy	Specifies an additional filtering strategy. At the moment, only the binary erosion filter is implemented
source_type	*camera_device* or *ros_topic*
image_topic	ROS topic on which source images are being published

```
class EventCameraEmulator(object):

    def get_events_image(image_1, image_2, ...):
        ...
    def get_events_image_rgb(image_1, image_2, ...):
        ...

    def compute_thresholded_diff_rgb_single_channel(image_1, image_2, theta, ...):
        ...
    def compute_thresholded_diff_rgb_multi_channel(image_1, image_2, theta, ...):
        ...
    def compute_thresholded_diff_salvatore_method(image_1, image_2, theta, ...):
        ...

    def get_visual_events_image(event_frame):
        ...
```

Listing 1: `EventCameraEmulator` Functions

SNN Simulator

The `snn_simulator` node receives event data and runs it through a simulated C-SNN to produce the neural activation maps required by the obstacle avoidance component[5].

We use the open-source, pytorch-based *BindsNET* python package[6] for running SNNs (presented in Hazan et al. (2018)). Through preliminary testing, we have determined this implementation to be adequate for the purposes of the presented approach. Among the package's features are efficient implementations of different spiking neuron models and learning rules from the literature, extensive SNN parameterization options, and a design that allows creating novel neuron models, learning rules, and architectures. In addition, "monitoring" tools provide access to data including spike trains running through the network and neuronal membrane potentials over time. In order to use the FST output representation described in section 4.1.4, we extended the implementation to incorporate the computation and recording of first spike times.

The `snn_simulator` node instantiates and configures a C-SNN according to user-specified parameters. Table 4.2 lists and describes the parameters that configure the `snn_simulator`. Upon receiving event images on the "/events_images" topic, the node processes the inputs by:

1. Optionally applying an event filtering strategy; we use binary erosion (see description in 4.1.2)
2. Downscaling the image (if necessary) to match the input layer size[7]
3. Assigning r_{ON} and r_{OFF} rate values (Hz) to ON and OFF events, respectively, for Poisson encoding

[5]Note that "simulation" here refers to the fact that the SNN is simulated in software as opposed to being run on a dedicated neuromorphic processor.

[6]https://github.com/BindsNET/bindsnet

[7]In our experiments, input images are of size 640x480 and are downscaled by a factor of ~ 3. The resultant image size often retains the salient event information while not being too computationally heavy to process; an important aspect for real-time operation.

```
std_msgs/Float32MultiArray fst_array
std_msgs/Float32MultiArray spike_counts_array
uint16 input_image_height
uint16 input_image_width
uint16 snn_sim_time
```

Figure 4.10: SNNOutput ROS message specification

```
std_msgs/Float32MultiArray fst_array
        std_msgs/MultiArrayLayout layout
                std_msgs/MultiArrayDimension[] dim
                        - string label
                          uint32 size
                          ...
        float32[] data
```

Figure 4.11: Float32MultiArray ROS message specification (from the std_msgs package)

(as presented in section 4.1.3)

4. Encoding the data into Poisson spike trains

The resulting spike data is then run through the C-SNN for a given simulation time period, T_{sim}, and network "monitors" are inspected to extract statistics of interest such as membrane potentials, spike counts, average firing rate, and the additional FSTs. For ease of transferring this information to the next component, we define a custom ROS message **SNNOutput**, which contains arrays of FSTs and spike count values in addition to meta-information, as shown in Figure 4.10. The float arrays can later be interpreted in terms of the correct neural activation map sizes by accessing the dimensions labeled in the **MultiArrayLayout** and **MultiArrayDimension** sub-fields of the **Float32MultiArray** sub-field message type (from the *std_msgs* package[8]), whose specification is shown in Figure 4.11). Finally, the **SNNOutput** message is published on the "/snn_output" topic.

Motion Controller

The `motion_controller` node encompasses the two components responsible for i) obstacle avoidance computations and ii) trajectory planning and control. It handles instantiating and configuring the DMP (effectively, the motion planner), running the motion loop in which velocity commands are executed by the PID controller to follow the trajectory plan, and adapting the trajectory plan on-line by receiving the SNN output data, decoding it into obstacle avoidance accelerations and adapting the DMP trajectory.

We use the open-source pydmps[9] package to run the DMP that computes the end-effector positions of the intended trajectory plan. Given a trajectory shape specification (pre-learned weights $\mathtt{w}_i$ of Equation

[8] http://wiki.ros.org/std_msgs
[9] https://github.com/studywolf/pydmps

Table 4.2: Parameters of the `snn_simulator` node

Parameter	Symbol	Description
sim_time	T_{sim}	Time period for which the SNN is run
snn_thresh	v_{thresh}	Membrane potential threshold at which a neuron spikes
snn_reset	v_{reset}	Membrane potential reset value following a spike
snn_rest	v_{rest}	Equilibrium (baseline) membrane potential value
snn_refrac	T_{refrac}	Refractory period
snn_decay	τ_{decay}	Time constant for membrane potential decay
snn_thresh_increase_at_spike	-	Factor by which v_{thresh} is increased at spikes
snn_thresh_decay	-	Time constant for potential threshold decay
init_weight_factor	w_c	Initial weight constant

4.5), initial and goal positions, and other parameters, the DMP steps through end-effector positions determined by the input parameters. The "step" function provides an "external_force" argument which supports an additive term that modifies the DMP equation and accordingly adapts the subsequent waypoint. This provides the functionality needed to incorporate obstacle avoidance adjustments to the planned trajectory (as described in section 4.1.5).

While the DMP generates a trajectory plan during execution, the `motion_controller` node listens to messages on the "/snn_output" topic and performs the decoding procedure needed to acquire the acceleration term, $\boldsymbol{\phi}$ (see section 4.1.4). In particular, `decode_snn_output` and `compute_potential_field` functions perform the extraction of "obstacle" points on the neural activation map and accordingly computing the potential field representation, respectively. The negative potential gradients result in $\boldsymbol{\phi}$, which is added, if non-zero, to the DMP "step" function. The ROS communication framework allows for the DMP execution and the reception (and processing) of SNN outputs to run simultaneously, ensuring that the motion is adapted with the latest possible information on sensed obstacles. In practice, we have found that augmenting the DMP with a decaying, running average of $\boldsymbol{\phi}$ (instead of instantaneous values) leads to smoother trajectories.

Within the DMP execution cycle, an inner motion loop handles the computation and application of velocity commands required to reach the next position in the trajectory plan. Here, we use our PID controller to drive the end-effector from the current position to the next. The twist values computed by the controller are then applied using functionalities from the *ros_kortex*[10] package. In particular, an implemented `KinovaArmController` class handles sending twist command requests to the Kinova arm *kortex_driver* (within *ros_kortex*) which in turn executes the low-level control commands that drive the motion. These requests are communicated through ROS services (as opposed to topic publishers and subscribers)[11]. The `KinovaArmController` instance (which provides other functionalities such as moving the arm to predefined joint and end-effector configurations) is the only component of the node and the

[10] https://github.com/Kinovarobotics/ros_kortex

[11] This relies on a SendTwistCommandHandler within the *kortex_driver* package, which was an experimental feature at the time of writing and was not activated by default in the original implementation.

pipeline that is specific to the Kinova arm. Therefore, the implementation described here can be adapted to other platforms or similar applications by simply integrating the associated low-level controller and adjusting parameters where necessary.

Apart from the aforementioned functionalities, the `motion_controller` node also enables visualizations and logging of parameters and metrics that facilitate debugging, execution monitoring and performance evaluation. Visual outputs include RViz visualizations of planned and adapted trajectories and obstacle avoidance vectors, as well as PF representations. In order to facilitate automated testing and evaluation (described in section 5.1.5) the node logs the values of all parameters, including those of the `event_image_streamer` and `snn_simulator` instances, in addition to quantitative metrics. The metrics logged after executions include trajectory length, execution time, the number of collisions, distance to the goal, and overall success.

Table 4.3 describes the parameters associated with the `motion_controller` node.

Table 4.3: Parameters of the `motion_controller` node

Parameter	Symbol	Description
goal_reaching_tol	$\delta_{\mathbf{g}}$	Goal reaching tolerance
position_reaching_tol	$\delta_{\mathbf{y}}$	Position reaching tolerance
obs_avoidance_dist_tol	δ_{obs}	Obs. avoidance tolerance
goal_reaching_timeout	-	Goal reaching timeout duration
position_reaching_timeout	-	Position reaching timeout duration
pf_method	-	PF method
p_0	p_0	Obstacle radius of influence (PF; Park)
eta	η	Constant gain (PF; Park)
grad_factor	C_δ	Gradient constant factor (PF; Park)
max_phi	ϕ_{max}	Maximum value of ϕ
aggregate_phi	-	Whether to aggregate ϕ values over time
phi_horizon	n_ϕ	ϕ history horizon
K_p	K_p	Proportional controller gain
K_i	K_i	Integral controller gain
K_d	K_d	Differential controller gain
motion_loop_frequency	f_v	Motion loop frequency
safety_strategy	-	Safety strategy
position_limits	-	Position limits
fst_activation_threshold_factor	t_{act}	FST activation threshold factor. "Obstacle points" are assigned at positions of neurons whose FSTs are before t_{act}

Gazebo Object Animator

For Gazebo simulation tests, a `gazebo_object_animator` node was developed to control the motions of dynamic obstacles in the environment for some evaluation tasks (see section 5.1.1). According to input parameters, the node animates the motion of a given object through a specified trajectory and enables triggering the motion by publishing a boolean message to a "\gazebo_object_animator\trigger" topic. During experiments, the `motion_controller` node is set to publish this trigger command as motion execution starts to induce a controlled, relative motion of the object(s). This is useful in conducting reproducible experiments when statistically evaluating the performance of the implementation or parameter sets, as it eliminates possible variances in results that may arise due to slightly varying testing conditions. Naturally, subsequent evaluations would involve removing this constraint to then test for the implementation's robustness to such variations, which are effectively unavoidable in real robot scenarios. The `gazebo_object_animator` node publishes to the "/gazebo/set_model_state" and "/gazebo/set_link_state" topics to control the positions of Gazebo objects.

4.2.2 Simulation

The implementation presented here is first evaluated in simulation before incorporating the obstacle avoidance module in real robot task executions. We use an existing, open-source Gazebo simulation of the Kinova Gen3 arm, and extend it to include an on-board RGB camera from which visual data can be obtained. The simulation integrates with the same control drivers and other components that run on the real robot, which are accessible through the *ros_kortex* ROS package.

Starting the development cycle in simulation has several advantages. In general, simulation enables more rapid testing and development by eliminating overhead in setting up the physical robot and its environment, etc. This also relates to removing concerns of safety (of people, the robot, and its surroundings) in early stages of development. In addition, the simulation allows for conducting large numbers of trials without concerns of wear-and-tear, power consumption, time, etc., as well as automating the execution and evaluation metric computation (as is done in the present work), greatly accelerating iterative development. The ability to largely control the environment is also beneficial in conducting directed tests of performance; by simulating custom environments with elements such as visual distractors, we can evaluate our implementation in different operating conditions and identify limitations.

ros_kortex is a ROS package built on the official Kortex API for interacting with Kinova robots and contains the Gazebo simulation in a *kortex_gazebo* sub-package. We extend the Gazebo simulation of the 7-DOF Gen3 arm with an Intel RealSense camera mounted at the top of the end-effector (see Figure 4.12). This is achieved by adding the official URDF descriptions of the Realsense camera available in the *realsense-ros* package[12] to the robot URDF specifications. In addition, we incorporate a Gazebo Realsense plugin that enables configuring data capture and accessing the camera data through ROS

[12]https://github.com/IntelRealSense/realsense-ros

topics[13]. (The aforementioned extensions can be found in a fork of the *ros_kortex* repository[14].) With the RGB data from the simulation on-board camera published in a "/camera/color/image_raw" topic, the `event_image_streamer` can be used to emulate events as though from a camera on the real robot.

The Gazebo simulation provides opportunities for incorporating different backgrounds, objects, and general environment setups. We utilize this in creating different testing scenarios that are conducive to a thorough analysis of the performance of the proposed approach. For example, by varying the types of objects representing obstacles and their textures in addition to the test case (static obstacle, dynamic obstacle, etc.) and backgrounds, we create different Gazebo "worlds" for tuning and testing the implementation (more details are presented in chapter 5). This is possible by customizing the XML files that launch the Gazebo simulation.

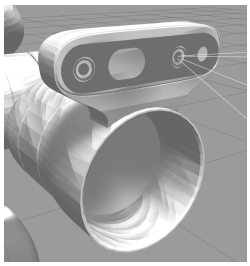


Figure 4.12: The Intel Realsense camera added to the Kinova simulation end-effector

4.3 Concluding Remarks

The approach and its implementation presented in this chapter enables obstacle avoidance on a camera-equipped manipulator through a novel method of utilizing event-based vision, spiking neural networks, and an adaptive trajectory representation for integrating high-level planning and low-level reactive motion corrections in reaching tasks. Its application provides not only the opportunity for conducting studies on obstacle avoidance in the domain of manipulation (which is less prevalent in the literature, as expressed in section 2.4), but to also explore a neuromorphic approach to the problem. Unlike in other works mentioned in section 2.4, this approach does not require a priori information on the environment nor obstacles (such as geometric shapes or CAD models).

We end this chapter by addressing some inherent limitations of the SNN-based obstacle avoidance module's current design, some of which have been mentioned in the delimitations of this thesis (see section 1.2). Intuitively, the module depends on the presence of motion, either of the robot or the obstacles, to perform as intended, since the generation of events depends on relative motion. However, this is expected to bear no adverse effect on the objective of obstacle avoidance, since collisions normally do not occur, and are thus not a concern, without motion. The current design does not take collisions of other robot links with an avoided obstacle into account, since the objective has been simplified to the avoidance of the end-effector. A holistic treatment of the robot's body is left for future work. Using a mounted camera for sensing places a clear constraint: the robot can not handle incoming collisions that are not perceivable within the camera's FOV. This stems from the choice of not deploying fixed cameras in the workspace (as some works reviewed in section 2.4 do), for example, since i) the approach would be less general and convenient with added requirements of setting up and calibrating a dedicated camera for every environment the robot is deployed in, and ii) event cameras' dependence on motion, which is naturally present in most tasks if the camera is mounted on the robot.

[13]We utilize an existing implementation for this, available in https://github.com/rickstaa/realsense-ros/blob/development-gazebo/realsense2_description/urdf/_d435.gazebo.xacro

[14]https://github.com/AhmedFaisal95/ros_kortex/tree/kinova-obs-avoidance-features

5

Evaluation Methodology

In this chapter, we present the procedure with which the neuromorphic approach to manipulator obstacle avoidance presented in chapter 4 is systematically evaluated. Our evaluation involves conducting experiments first in a simulated and then on a real Kinova Gen3 robot. We exploit simulations to facilitate the initial stages of development and parameter tuning, particularly in minimizing potential safety risks, expended human time and effort, excessive wear and tear, and other physical consequences associated with conducting a large number trials on a robot. Perhaps most importantly for a statistically thorough evaluation, the simulation allows for a degree of test condition reproducibility that is difficult to achieve in the physical world. By possessing more minute control over test environments and variables, we can eliminate undesirable variations in conditions and therefore be able to draw reasonably certain conclusions on several aspects of performance from batches of simulated trials. These controlled test scenarios are useful for evaluating the performance of the pipeline (end-to-end) and isolated components. Ultimately, the purpose of the experiments is to compare the outcomes of task executions with and without the presented SNN-based obstacle avoidance module, with the aim of verifying the merits of our approach.

Following extensive evaluations in simulation, we run similar experiments on a real robot platform for a final validation of the proposed approach. This is an integral aspect of this thesis: our approach has been developed to work in real scenarios, as opposed to a purely simulation-based proof-of-concept. These experiments are intended to explore the general viability of the event-based, SNN-based obstacle avoidance module in a real setting and to assess how well the implementation and parameters optimized in simulation transfer to the real world.

We evaluate performance through a set of formalized metrics and appropriate data visualizations. Quantitative metrics that include numbers of collisions, trajectory lengths, velocity magnitudes, and general success rates are defined to aid in critically assessing the various parameterizations and the final "model". These metrics are accompanied by 3D visualizations of pre-planned and adapted trajectories, distributions of velocity and acceleration values, and other visual illustrations. By automating the execution of simulated trials, the computations of metrics, and generation of supplementary plots, we develop an efficient testing, evaluation, and iterative development procedure. In addition, qualitative metrics such as predictability, safety, and reliability are used to further describe trajectory properties.

Section 5.1 encompasses all details of the simulation experiments, beginning with the design of the tasks and the tuning $\rightarrow$ validation $\rightarrow$ testing procedure, which encompass the development cycle and

the final evaluation experiments. This is followed by a detailed break-down of how experiment trials are conducted, the quantitative and qualitative evaluation criteria and visualizations, and automation details. Section 5.2 deals with how our approach is evaluated in real robot experiments and follows a similar outline, containing descriptions of the evaluation tasks, experiment procedure, and evaluation metrics.

5.1 Simulation Experiments

As explained in section 4.2.2, we employ an adaptation of the Gazebo simulation of the Kinova Gen3 arm provided in the *ros_kortex* software package. By controlling a variety of environmental variables such as the background and obstacle properties, we instantiate different scenarios within a set of simple reaching tasks to conduct experiments designed to evaluate obstacle avoidance behaviour. These are used in a tuning → validation → testing procedure, where three sets of scenarios are designated for initial parameter tuning, parameter set validation and selection, and experiments for the final assessment of a selected parameter set, respectively. This procedure is intended to explore how generalizable the presented approach is to different conditions and to identify and analyze its limitations. Experimental trial executions, data visualizations, and quantitative metric computation are handled by an automated evaluation implementation. These results, in addition to a set of qualitative criteria determined from the data collected during experiments, are used to critically evaluate parameter sets and the implementation of the proposed approach as a whole.

All simulation experiments are conducted on an HP Omen 15 laptop running the Linux Ubuntu 20.04 operating system. The system contains an octa-core AMD Ryzen 7 4800H processor, 16GB of RAM, and an NVIDIA GeForce GTX 1660 Ti graphics card.

5.1.1 Evaluation Tasks

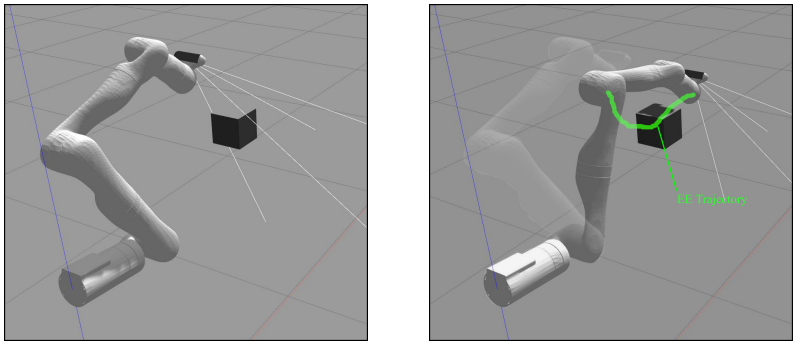
We formulate four simple goal-directed manipulation "tasks" to be performed by the Kinova arm during our experiments. These tasks are designed to aid in demonstrating and evaluating obstacle avoidance capabilities. In each task, the robot's regular course of action (a pre-planned motion trajectory or maintaining a set position) would lead to imminent collision, unless a deliberate avoidance action is taken to appropriately adapt. Therefore, we address situations in which a naïve motion planning and control approach that does not consider unexpected obstacles during execution is certain to fail, and investigate if and how well our approach to obstacle-aware manipulation trajectories solves the problem. Apart from the manipulator's avoidance ability, these tasks also enable us to study characteristics of avoidance behaviours and the arm's motion in general. We cover instances of both static and dynamic obstacles in the set of tasks, as the circumstances are significantly different and can provide different insights on the strengths and limitations of the proposed approach.

The four tasks we base our simulation experiments on are described in the following:

- **Task 1**: The arm must reach a commanded goal position that lies behind a **static** obstacle, which is positioned between the arm and the goal such that a nominal motion trajectory would lead the end-effector to collide with the obstacle.

- **Task 2**: The arm must reach a commanded goal position as a **dynamic** obstacle enters the field of view and crosses the end-effector's path; the trajectory of the obstacle is set to intersect with a nominal motion trajectory and thus lead to a collision.
- **Task 3**: The arm must reach for an object on a table that is partially occluded by a **static** obstacle, which is positioned between the arm and the goal such that a nominal motion trajectory would lead the end-effector to collide with the obstacle.
- **Task 4**: The arm must maintain its initial position (set-point) as a **dynamic** obstacle moves directly towards it; the trajectory of the obstacle is set to intersect with the initial end-effector position and thus lead to a collision, and the arm is free to move but must constantly minimize distance to the initial position.

Figures 5.1–5.4 illustrate the setups of the tasks in Gazebo. Each figure depicts the start and end of each task, in addition to a baseline (nominal) end-effector trajectory that does not consider obstacles.

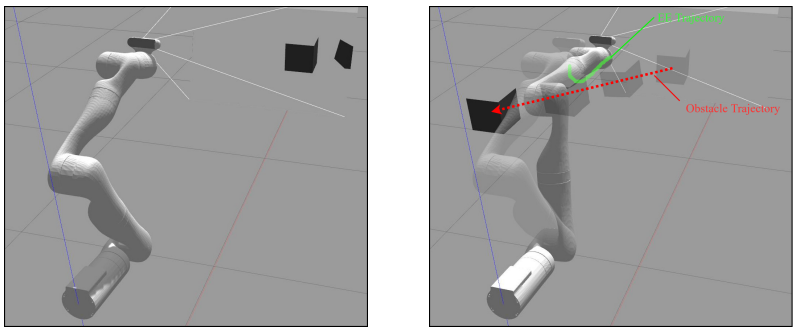


(a) Start (b) End

Figure 5.1: Task 1: reaching for a goal position behind a static obstacle. A nominal end-effector trajectory is illustrated in green.

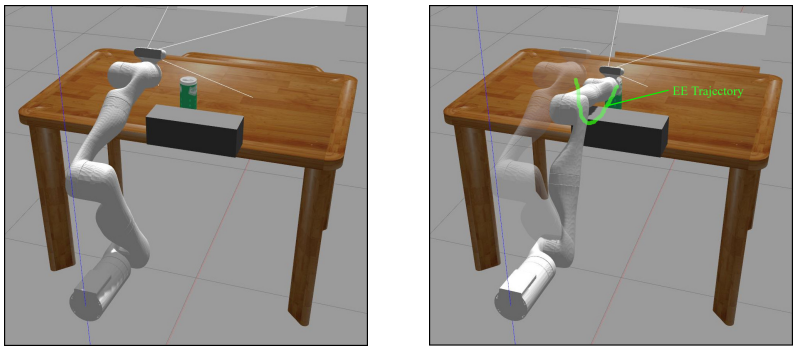
In all tasks, the arm always starts at the same initial position and is commanded to reach the same goal position. This is not a necessity for our implementation, but is enforced for the uniformity of experiments and comparability of trial results. The goal position is always within the camera's FOV, regardless of whether it is occluded. For the purpose of the experiments, the initial joint configuration of the arm is immaterial, insofar as it would not obstruct nominal trajectories to the goal, or reasonably-bounded trajectory adaptations thereof, due to, for e.g., singular configurations or self-collisions[1]. We use two different arm configurations: the default configuration of the *ros_kortex* simulation in initial development stages (Figure 5.5a) and a configuration designed to mimic that of an existing robot platform on which experiments are conducted (Figure 5.5b)) for the tuning → validation → testing procedure. This is not expected to adversely affect the final results nor the derived conclusions, but provides some validation of

[1]Note that self-collisions are not a concern in of themselves due to inherent safety checks on the Kinova arm that prevent motions leading to self-collisions. Rather, the concern is not approaching configurations that would necessitate the activation of these safety measures.



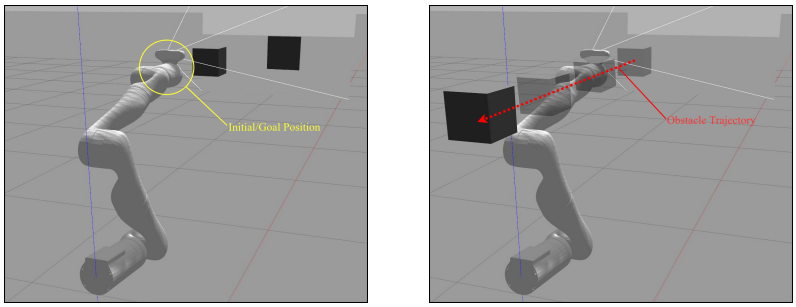
(a) Start (b) End

Figure 5.2: Task 2: reaching for a goal position in the presence of an interfering dynamic obstacle. The obstacle's trajectory and a nominal end-effector trajectory are illustrated in red and green, respectively.



(a) Start (b) End

Figure 5.3: Task 3: reaching for a goal position on a table behind a static obstacle. A nominal end-effector trajectory is illustrated in green.

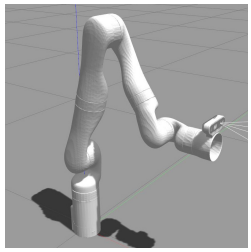


(a) Start (b) End

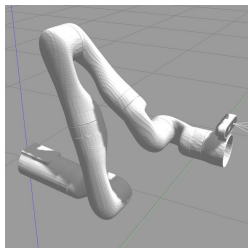
Figure 5.4: Task 4: maintaining a set position as a dynamic obstacle moves towards the end-effector. The obstacle's trajectory and a nominal end-effector trajectory are illustrated in red and green, respectively.

the independence on arm configuration. In tasks 1, 2, and 4, the obstacle is no larger than a $0.1m^3$ cube, while the static obstacle in task 3 is a cuboid with dimensions {0.1, 0.3, 0.1}.

We vary a set of environmental variables in order to instantiate different testing scenarios for each task. These variables include the background, obstacle type, obstacle color, and obstacle speed. Table 5.1 outlines the variables we designate for each task and the values that they can take. Here, we define a "scenario" as an instance of a task characterized by a unique set of variable values. As an example, the cells highlighted in gray on Table 5.1 define a scenario where the objective is to accomplish task 4 in the *Bookstore* environment (dictating the background) for a *brick-pattern box* obstacle traveling at *medium* speed; this can be expressed with the set: {'Task 4', 'Bookstore', 'Box', 'Brick Pattern', 'Medium'}.



(a) Default configuration



(b) Experimental configuration

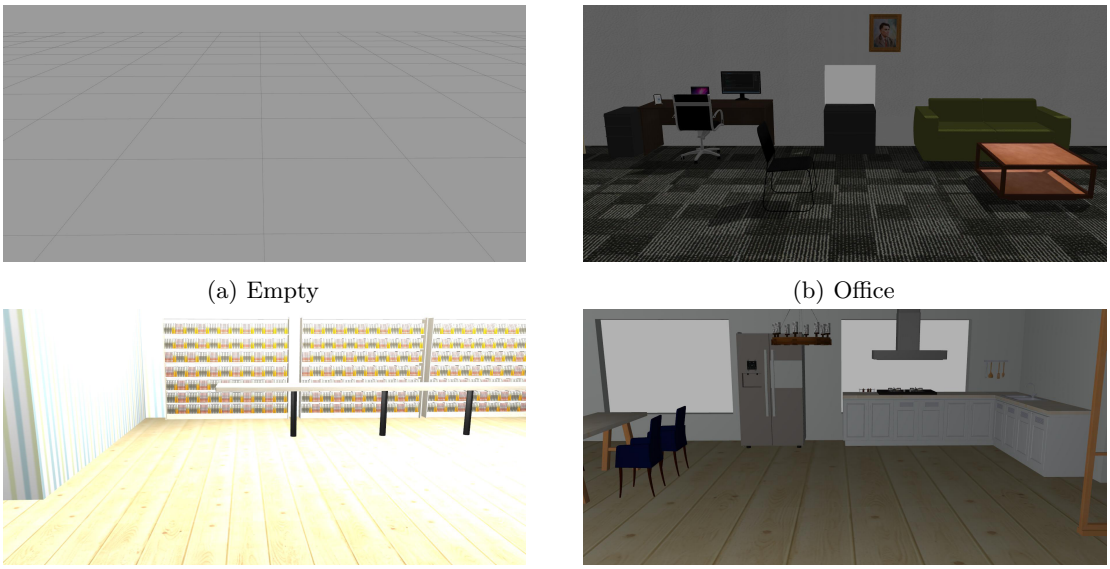
Figure 5.5: The Kinova Gen3 in the Gazebo simulation

Most task variables are expected to exert varying influences on the visual input to the obstacle avoidance module and thus its ultimate performance. An obstacle's type and color may induce non-negligible differences in event data and patterns. Background textures, colors and relative lighting could also have unexpected consequences that may induce undesirable event activity and potentially hinder the arm's accomplishment of the task. We term any such possibly detrimental background properties "visual distractors" and purposefully inject them in scenario backgrounds to investigate their effects. Furthermore, varying obstacle speed is expected to have an influence both on the visual saliency of events and how effective executed motions are. Since events are generated from relative motion through light intensity differences, motions that are slower or faster than certain thresholds may induce little to no event activity, and speed values in the intermediate range may have significantly varying effects on event output. With regards to motion execution, it is reasonable to assume that trajectory adaptations may be more or less capable of achieving avoidance depending on a dynamic obstacle's speed (with faster obstacles likely being more challenging), due to any latencies in motion correction and physical limitations. Therefore, the objective of testing different variable values is to examine whether these effects do manifest, how they affect the performance of our approach if they do, and the degree to which it can cope with the consequences, if any. The latter point is especially important for identifying limitations of selected parameter values and/or the approach as a whole. Finally, testing in different scenarios provides some validation of how applicable the obstacle avoidance behaviour is in different environmental conditions, before transferring to a real robot. Figures 5.6 and 5.7 depict the different values of the background, obstacle type, and obstacle color variables presented in Table 5.1.

The variety of scenarios we have access to through the combinations of the variables in each task is instrumental in the systematic parameter set selection and testing procedure described next.

Table 5.1: Task variables that define experimental scenarios.
Example scenario (gray cells): {Task 4, Bookstore, Brick pattern, Box, Medium}.
Approximate obstacle speeds: Low = $0.09m/s$, Medium = $0.17m/s$, High = $0.36m/s$.

Task	Variable	Values			
Task 1	Background	Empty	Office	Bookstore	Kitchen
	Obstacle Type	Box	Buckyball	Spiky Sphere	Rock
	Obstacle Color	White	Red	Yellow-Black	Brick Pattern
Task 2	Background	Empty	Office	Bookstore	Kitchen
	Obstacle Type	Box	Buckyball	Spiky Sphere	Rock
	Obstacle Color	White	Red	Yellow-Black	Brick Pattern
	Obstacle Speed	Low	Medium	High	
Task 3	Background	Empty	Office	Bookstore	Kitchen
	Obstacle Color	White	Red	Yellow-Black	Brick Pattern
Task 4	Background	Empty	Bookstore	Kitchen	
	Obstacle Type	Box	Buckyball	Spiky Sphere	Rock
	Obstacle Color	White	Red	Yellow-Black	Brick Pattern
	Obstacle Speed	Low	Medium	High	



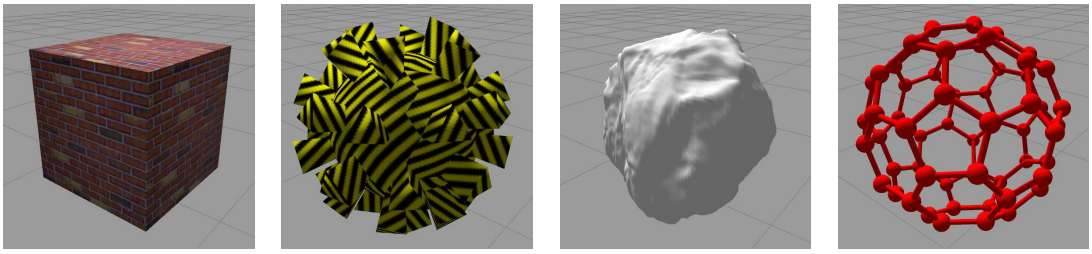
(a) Empty

(b) Office

(c) Store (high illumination)

(d) Kitchen (low illumination)

Figure 5.6: Different backgrounds (Gazebo worlds) used in task scenarios, listed in Table 5.1



(a) Box: Brick Pattern (b) Spiky Sphere: Yellow-Black Pattern (c) Rock: White (d) Buckyball: Red

Figure 5.7: Different obstacle types used in task scenarios, listed in Table 5.1. Each obstacle also illustrates one of the possible obstacle colors.

5.1.2 Tuning, Validation and Testing Procedure

In pursuing a principled approach to parameter tuning and selection in addition to the final evaluation, we draw insights from a methodology often employed in machine learning (ML) research. ML practitioners commonly optimize ML models on a "training" set of examples and evaluate a chosen model on a "testing" set of unseen examples, in order to rule out biases to the training examples on the final evaluation of the model. When it is necessary to choose from different models and hyperparameter[2] values, a separate "validation" set of examples is often used to evaluate trained models on examples that have not been observed during training to estimate how well they generalize and to select a model or set of hyperparameter values to run on the testing set. This training, validation, and testing strategy represents a systematic procedure for the optimization and unbiased evaluation of a given system. Although we do not train a model in the current implementation of our proposed approach, we follow a similar strategy for optimizing and evaluating our neuromorphic obstacle avoidance module.

In this work, our implementation and parameter selection are evaluated based on performance in different task scenarios. We treat these scenarios as analogous to the "examples" presented to an ML model, and sample from the set of possible scenarios to construct disjoint sets of different scenarios drawn from the same distribution (as defined in Table 5.1). On this principle, we formulate sets of task scenarios that fulfill the following purposes:

- Tuning: a small set of scenarios chosen for manually tuning parameters and instantiating candidate parameter sets.
- Validation: a medium-sized set with more task variable variations used for evaluating tuned parameter sets.
- Testing: a larger set of randomly-selected scenarios for evaluating a chosen parameter set, which

[2]In ML research, hyperparameters describe the parameters that configure the model to be trained, as opposed to internal parameters that are tuned by the learning algorithm during training. Examples include the number of training examples and the learning rates, as opposed to the tunable weights of a neural network model.

contains even more task variable variations and unobserved variable values.

With these scenario sets, we optimize and evaluate our approach through a tuning, validation, and testing strategy[3] (depicted on Figure 5.8). During development and parameter tuning, the *tuning set* is used to tune parameters to achieve adequate performance, while the *validation set* is used to evaluate the degree to which a given "model" generalizes and to select between multiple candidates. Our formal experiments involve running the final implementation and parameter set on the *testing set* of scenarios and comparing to executions that do not utilize the SNN-based obstacle avoidance module.

The tunable parameters that we consider in this procedure are those that govern the behaviour of each pipeline component outlined in section 4.2.1: the event image streamer, SNN simulator, and motion controller nodes (see tables 4.1, 4.2, and 4.3). A specific set of values of these parameters may represent an instance of a "model", whose performance is to be compared to other sets. We also consider additional modifications to component implementations to improve performance, if any, as part of a given parameter set. In other words, these modifications are associated to a parameter set describing a "model" instance.

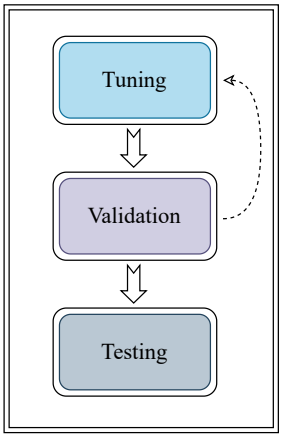


Figure 5.8: A depiction of the proposed tuning → validation → testing procedure.

During the tuning phase, the instantiation of parameter sets to be iteratively optimized can be arbitrary or follow a directed approach. In particular, the validation phase can be used to evaluate tuned parameter sets and inform the subsequent creation of new parameter sets to be further refined on the tuning set in a feedback loop (depicted by the dotted arrow on Figure 5.8). As an example, one could combine the characteristics of parameter sets that perform best on the validation set in a given iteration to create a new derivative parameter set. The method of deriving such a combination may be manual or involve a learning or optimization algorithm such as evolutionary optimization, SMAC[4] or some form of reinforcement learning. Here, we perform this procedure manually, but we intend to explore an automated optimization procedure in future extensions.

From the task variables and their values defined in table 5.1, we can instantiate a total of 416 scenarios. We choose to allocate 3 in the tuning set, 8 in the validation set, and 20 in the testing set. The tuning and validation scenarios are manually selected in order to guarantee some variation in scenario properties (with the validation set containing more variations, some unobserved during tuning) but also to ensure that at least one task and one background is never observed in either phase. Testing scenarios are sampled randomly from the remaining pool of scenarios, with the only restriction being at least N_s scenarios per task (including the unobserved task). We set $N_s = 5$. Therefore, the testing set is composed of novel scenarios that include a novel task and a background/environment. In this case, these were task 2 and the Kitchen environment (which is characterized by a dimmer ambient lighting than the other environments).

Table 5.2 lists the selected tuning and validation scenarios and randomly sampled testing scenarios. Note the inclusion of an additional scenario at the top (Task 1, Empty, Cracker Box). This scenario is

[3]Note that we name the first phase "tuning", not "training", to make explicit the fact that we currently optimize by manually tuning the different components' parameters, as opposed to running a learning algorithm.

[4]Sequential Model-based Algorithm Configuration

used to establish baseline parameter values in the first tests of the implementation, termed a pre-tuning phase (see details in chapter 6).

5.1.3 Evaluation Metrics and Criteria

Establishing a set of metrics and criteria based on which the proposed approach can be systematically assessed and improved is a vital aspect of a thorough evaluation procedure. The review of the literature presented in section 2 has highlighted a scarcity in reported results of obstacle avoidance performance and quantitative metrics thereof, which in general hinders methodical comparisons to other related approaches and an objective evaluation of the proposed approach. This may be attributed to the general difficulty of evaluating obstacle avoidance methods expressed in Minguez et al. (2016), where the lack of a metric to quantitatively measure and compare performances is put forth as a primary cause. We attempt to address this lack by defining and grounding a set of quantitative metrics and qualitative criteria for our evaluation.

Quantitative Metrics

Table 5.3 lists the quantitative metrics we use to analyze and compare the performances of different parameter sets and to ultimately evaluate the proposed approach during the testing phase. These metrics are designed to holistically evaluate the performance of the SNN-based obstacle avoidance module. In addition, they enable comparisons to task executions that do not utilize the obstacle avoidance module on the basis of task success and along other dimensions that describe trajectory properties. The task scenarios defined in section 5.1.2 provide situations in which the performance of obstacle avoidance behaviours and characteristics of the arm's motion in general can be tested, and the metrics provide the tools required to quantitatively analyze the resultant data. The metrics are discussed in further detail in the following.

The execution time, T, simply refers to how long it takes to execute the task. Relative to a nominal trajectory[5], it indicates how much more time is expended on average in obstacle avoidance maneuvers. The trajectory length estimates the distance traveled by the end-effector, and is similarly most interesting in comparing to nominal trajectories to assess how far obstacle avoiding trajectory adaptations tend to deviate. The length can be estimated by summing the euclidean distances between consecutive measurements of end-effector position during execution:

$$l_{\mathbf{Y}} = \sum_{i=1}^{N_{pos}} ||\mathbf{y}_i - \mathbf{y}_{i-1}||_2 \tag{5.1}$$

Here, N_{pos} refers to the number of recorded positions visited by the end-effector while executing the trajectory, $\mathbf{Y}$. It is generally desirable to minimize both elapsed time and traveled distance as a secondary objective to successful obstacle avoidance and reaching the goal, In simulation, the number of collisions, $N_{collisions}$ represents a count of the instances in which the end-effector intersects an obstacle in 3D space (provided that physical collision effects are disabled in the simulation). Intuitively, this provides an

[5]The term "nominal trajectory" here is used to refer to the trajectory that would have been executed by the arm in each task if there was no obstacle or if it was not actively in consideration (i.e. with no obstacle avoidance module).

Table 5.2: List of tuning, validation, and testing scenarios

Scenario ID	Set	Scenario Specification
0	Pre-Tuning	Task 1, Empty, Cracker box
1	Tuning	Task 1, Store, Red Box
2	Tuning	Task 4, Store, Brick, Rock, Medium Speed
3	Tuning	Task 3, Empty, Brick
4	Validation	Task 1, Office, Red, Buckyball
5	Validation	Task 1, Office, Yellow-Black, Rock
6	Validation	Task 1, Empty, Brick, Spiky Sphere
7	Validation	Task 4, Empty, Yellow-Black, Box, Medium Speed
8	Validation	Task 4, Office, Yellow-Black, Buckyball, High Speed
9	Validation	Task 4, Store, White, Spiky Sphere, Low Speed
10	Validation	Task 3, Office, White
11	Validation	Task 3, Store, Yellow-Black
12	Testing	Task 1, Empty, Yellow-Black, Spiky Sphere
13	Testing	Task 1, Store, Brick, Buckyball
14	Testing	Task 1, Kitchen, White, Buckyball
15	Testing	Task 1, Empty, Yellow-Black, Box
16	Testing	Task 1, Store, Yellow-Black, Rock
17	Testing	Task 2, Office, Red, Buckyball, High Speed
18	Testing	Task 2, Office, Red, Rock, Medium Speed
19	Testing	Task 2, Store, Brick, Box, Medium Speed
20	Testing	Task 2, Kitchen, Yellow-black, Box, High Speed
21	Testing	Task 2, Empty, Red, Spiky Sphere, Medium Speed
22	Testing	Task 3, Empty, Red
23	Testing	Task 3, Empty, Yellow-Black
24	Testing	Task 3, Kitchen, White
25	Testing	Task 3, Kitchen, Red
26	Testing	Task 3, Store, Red
27	Testing	Task 4, Store, Yellow-Black, Buckyball, Low Speed
28	Testing	Task 4, Store, Red, Buckyball, High Speed
29	Testing	Task 4, Empty, Yellow-Black, Spiky Sphere, Medium Speed
30	Testing	Task 4, Empty, Yellow-Black, Box, Low Speed
31	Testing	Task 4, Empty, Brick, Buckyball, Low Speed

Table 5.3: Quantitative Performance Metrics

Metric	Symbol	Description
Execution Time	T	Time required to complete the task
Trajectory Length	$l_{\mathbf{Y}}$	Distance traveled by the end-effector
Number of Collisions	$N_{collisions}$	No. of instances in which the EE intersects an obstacle (in simulation)
Distance to Goal	d_G	Euclidean distance to the goal at the end of execution
Success	S	True **iff** no collisions **and** distance to goal $< \epsilon$
Velocities	$\dot{\mathbf{y}}$	Magnitudes of instantaneous end-effector velocities
Accelerations	$\ddot{\mathbf{y}}$	Magnitudes of instantaneous end-effector accelerations
Computation Time	T_{comp}	Average time elapsing between event data transmission and trajectory adaptation

indication of how often the end-effector collides with obstacles and the degree to which collisions are sustained or eventually adapted for[6].

The distance-to-goal metric measures how well a given execution fulfills the goal reaching aspect of the task and is obtained from the euclidean distance of the end-effector's final position, y_T, to the commanded goal position, g:

$$d_G = ||\mathbf{y}_T - \mathbf{g}||_2 \tag{5.2}$$

The success of an execution is captured in the binary *success* metric, which incorporates the number of collisions and distance-to-goal metrics: in order to be considered a success, a given trial must i) involve no collisions and ii) end with the end-effector close to the goal, according to a goal-reaching tolerance $\delta_{\mathbf{g}}$:

$$success = \begin{cases} 1, & \text{if } N_{collisions} = 0 \text{ and } d_G < \delta_{\mathbf{g}} \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

We also measure the magnitudes of instantaneous end-effector velocities and accelerations during executions. These provide important insights on motion properties of adapted trajectories, relative to nominal executions, particularly with regards to qualitative criteria that include the safety and predictability of obstacle avoiding motions. Finally, the computation time metric measures the average time it takes to compute and apply obstacle avoidance trajectory adaptations, from the input event (or RGB) data to the output velocity commands. This metric evaluates the algorithmic performance of the implementation, particularly regarding suitability to real-time operation, and provides a comparison to other works that perform a similar analysis of computation time (such as Mronga et al. (2020)).

[6]Note that, strictly speaking, the average number of recorded collision instances will depend on the frequency at which this measurement is made; if the time between measurements in run A is shorter than in run B, an identical time period during which the end-effector intersects an obstacle will generate more collision instances in A than in B. We avoid this obvious error by ensuring that the measurement frequency is the same across all execution runs we conduct, such that the absolute numbers of collisions are unimportant, and the relative numbers provide a fair comparison of how often the arm is in collision.

Qualitative Criteria

Table 5.4: Qualitative Performance Criteria

Criterion	How to Evaluate
Reliability	Ratio of successful trials to total no. of executions in imminent collision cases
Predictability	Frequency and magnitudes of changes in direction; low angular velocities indicate fewer changes in direction i.e. more predictability
Safety	Magnitudes of velocities and accelerations
Naturalness	i) Magnitudes of jerk, a measure of trajectory smoothness ii) Visual observations of trajectory shapes

Table 5.4 contains a list of qualitative criteria we propose to describe properties of the trajectories produced by the obstacle avoidance module as well as to estimate how much their characteristics deviate from the nominal. Ideally, adapted trajectories would succeed in avoiding obstacles while remaining qualitatively similar to nominal trajectories. These criteria can be subjectively assessed purely from visual observations during experiment executions. However, in order to conduct a formal qualitative evaluation, we ground each of the criteria in terms of quantitative measures, where possible, aiming at more objective interpretations. In the following, we define each of the criteria and how they are evaluated.

The *Reliability* of the proposed solution can be defined as the degree to which it produces consistently positive results in successive trials under reasonably consistent conditions. We expect a reliable system to repeatedly achieve similarly positive results, as opposed to significant variances and/or failure rates, provided that the task conditions do not substantially differ from the system's intended operating environment. This is related to, but not the same as, repeatability, which characterizes results that are consistent throughout repetitions, but contains no notion of the degree to which these results are positive. One way to estimate the reliability of the SNN-based obstacle avoidance module is through the ratio of successful trials (according to our success metric) to the total number of trials in imminent collision situations. The task scenarios defined in section 5.1.1 provide the imminent collision situations in consistent conditions required to test for this.

Predictability describes to what extent the system's behaviour can be known or predicted in advance. The behaviour of a system described as predictable can be estimated by observation with a relatively high degree of certainty, regardless of success or performance. While a variety of aspects could be suggested for deciding on motion predictability, we constrain our attention to one: changes in direction. The frequency or magnitude of changes in direction within a trajectory affect its predictability: the results of a motion controller that produces fewer directional changes are easier to intuit or predict from historical observations (within or across executions). We can quantify the local degree of direction change at every position along the end-effector's trajectory from the angle between the vectors formed by connecting that positional point to the previous and the next point, respectively, as is done in Salarpour & Khotanlou (2019). Given trajectory positions $\mathbf{y}_k$, $\mathbf{y}_{k-1}$, and $\mathbf{y}_{k+1}$, and resultant vectors $\mathbf{v}_1 = (\mathbf{y}_k - \mathbf{y}_{k-1})$ and $\mathbf{v}_2 = (\mathbf{y}_{k+1} - \mathbf{y}_k)$,

the direction change, ζ, at y_k can be calculated from the dot product:

$$\zeta = \arccos\left(\frac{v_1 \cdot v_2}{|v_1||v_2|}\right) \tag{5.4}$$

We are particularly interested in the derivative, $\dot{\zeta}$, as the predictability measure. The magnitudes of ζ provide an indication of path curvature, and higher values do not necessarily indicate unpredictable motions[7]. On the other hand, the magnitudes of $\dot{\zeta}$, which could be expressed as an angular velocity, are related to how often and how drastically the end-effector changes the direction in which it is moving. When these are often and high in magnitude, they are likely to indicate a trajectory with sudden directional changes which can be described as unpredictable. We therefore utilize this angular velocity metric to assess predictability.

In general, *safety* refers to the degree to which the system is unlikely to cause danger, risk, or injury. In the case of the manipulation tasks addressed here, we can examine the magnitudes of applied velocities and accelerations of the end-effector as a measure of safety. Intuitively, higher velocities tend to increase the risk of collisions with other objects or people in the environment, and the magnitude of potential damage. We also examine the magnitudes of instantaneous accelerations, though velocities are more often used in safety assessments.

The final criterion we attempt to evaluate for is *naturalness*, which qualifies how natural i.e. normal, expected, ordinary, or logical[8] an object in question is. To the average observer, the degree to which a manipulator's trajectory is natural involves how closely it resembles motions produced by a human, for example. This is in part dictated by the trajectory smoothness, which can be quantified through jerk: the derivative of acceleration. As described in Gasparetto et al. (2015), the optimization of manipulation trajectories for minimum-jerk is a common approach particularly for obtaining smoother motions, and is inspired by studies indicating that human arm motions are optimized for the same criterion[9] (see also Simon & Isik (1993), Fligge et al. (2012)). The latter fact further supports the magnitudes of jerk as a potential indication of trajectory naturalness.

The qualitative criteria discussed in this section are assessed during the final experiments run on the testing set. This is in contrast to the quantitative metrics, which are instrumental in selecting, evaluating, and comparing parameter sets during tuning and validation, as well as in the testing phase.

Qualitative Evaluation Visualizations

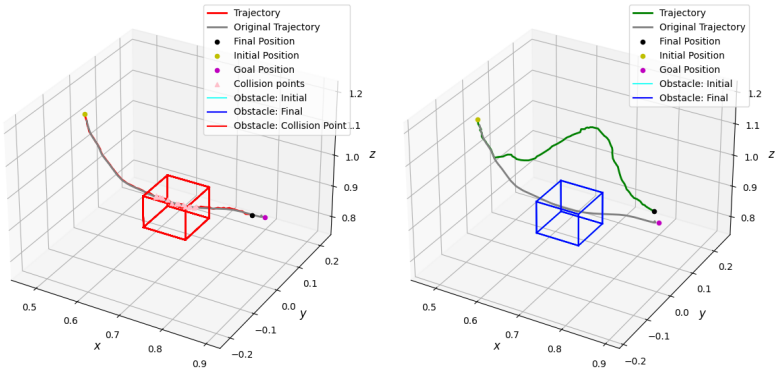
In order to visualize trajectory executions, we utilize annotated three-dimensional spatial plots of end-effector positions.

Figure 5.9 illustrates a plot of two Task 1 executions, the first with a pre-planned trajectory and the second involving the obstacle avoidance module. Trajectories are color-coded: the original (pre-planned)

[7] As an example, a sequence of high, but relatively similar ζ values are characteristic of a curving trajectory with a potentially small radius of curvature but a constant turning motion. In fact, the similarity of subsequent ζ values indicates that a subsequent portion of the motion is relatively easy to predict from an earlier portion.

[8] This is derived from a definition in the Collins English Dictionary

[9] As an additional side-note, the less smooth robot arm motions are, the more potentially damaging they are to the robot's actuators as well.



(a) Without obstacle avoidance (b) With obstacle avoidance

Figure 5.9: 3D trajectory plots for two Task 1 executions. The box depicts the obstacle to be avoided.

trajectory in gray, a successful trajectory in green, and a failed trajectory in red. An obstacle is visualized as a blue cuboid, and instances of collisions are denoted with pink triangles and red obstacle silhouettes at collision points. The plots are generated following executions in an interactive graphical interface, and are convenient in examining the trajectories and determining results at a coarse level of detail.

5.1.4 Experiment Procedure

In this section, we outline the details of the procedure we follow to conduct our experiments , following the selection of a parameter set in the tuning and validation phases. These experiments involve running N_{trials} executions in all task scenarios within the testing set (defined in section 5.1.2) in two cases:

1. Nominal executions that do not involve the obstacle avoidance module (the benchmark)
2. Executions that incorporate the SNN-based obstacle avoidance module, using the selected parameter set

In our experiments, we run $N_{trials} = 40$ trials in both cases for each scenario. From the data recorded from these batches of executions, we then analyse the obstacle avoidance performance using the evaluation metrics and criteria described in the previous section, particularly in comparison to the nominal executions.

A single run of our experiments in a given task scenario involves the following steps:

1. Start the Gazebo simulation, configured for the given task scenario, by running the `spawn_kortex_robot` ROS launch file of the `kortex_gazebo` package, parameterized by the appropriate configuration file.
2. Start the event camera emulation and SNN simulation components by running the `event_image_streamer` and `snn_simulator` nodes with the appropriate configuration files.
3. If the scenario involves a dynamic object, start the Gazebo object animator component by running the `gazebo_object_animator` node with the appropriate configuration file.

4. Move the arm to the initial configuration (depends on the task)[10].
5. Command the arm to move to the goal position (depends on the task) by running the `motion_controller` node with the appropriate configuration file, and choosing whether to use SNN feedback (activating obstacle avoidance).
6. After task execution, record all relevant data and compute performance metrics.

In step 1, the Gazebo configuration is defined by a pre-set "*.world" configuration file which specifies the models and properties required to create the conditions of the given scenario, including the environment, objects and obstacles. Each scenario in Table 5.2 is defined by a configuration file. The ROS nodes constituting the SNN-based obstacle avoidance pipeline (`event_image_streamer`, `snn_simulator`, and `motion_controller`) are similarly all parameterized by YAML files listing their chosen hyper-parameters and/or some `roslaunch` parameters. These parameters form a "parameter set", as it is referred to throughout this section, which dictates the eventual obstacle avoidance behaviour. The `gazebo_object_animator` node controls the position of a given object in the simulation and was developed to execute pre-defined dynamic obstacle trajectories for controlled experiments. These trajectories are set to start and end at precise points in time, relative to the execution of the arm's trajectory, in order to create the imminent collision cases defined in tasks 2 and 4. This is implemented through ROS messages from the `motion_controller` that trigger the obstacle trajectory when execution starts. The trajectories are also defined in a YAML configuration file and their speed can be parameterized. Appendix B contains examples of the configuration files for each of the nodes described here.

The initial joint configuration of the arm has been chosen such that a wide range of obstacle avoidance trajectories are reasonably possible in the context of the defined tasks, with fewer risks of limiting or singular configurations. The arm's base position is set to match the robot platform used to run real-world experiments, shown in Figure 5.10a. This ensures that any interesting differences in observations between the simulated and real robots arise from factors other than the potential constraints imposed by a different arm joint configuration, since they would be more interesting in drawing insights about the performance of our approach (particularly in transferring to the real world). Figure 5.10b shows the arm in RViz, positioned in the initial joint configuration we specify for our experiments. Relative to the world frame, the "base_link" frame at the base of the arm is located at the pose: {0.0, 0.0, 0.75, 0.0, 1.7, 0.0}[11].

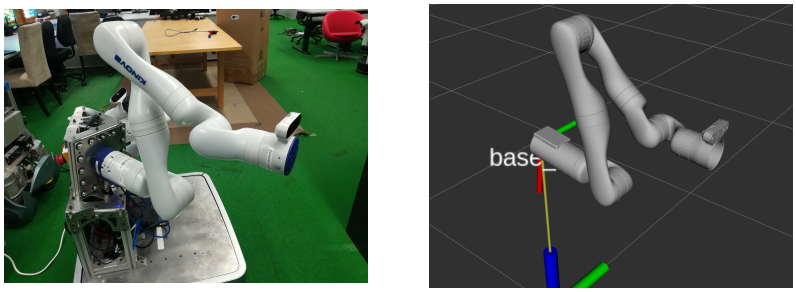
In each of the tasks, the arm is set to start in a configuration similar to the one just described but with different initial end-effector poses. Similarly, each task is characterized by a different goal position. Note that the orientation of the end-effector is not of concern in these experiments and is not explicitly controlled for[12]. Nevertheless, we report the initial orientation roll, pitch and yaw angles since they affect the field of view of the camera. Table 5.5 lists the initial and goal end-effector poses that are defined for every task, while Table 5.6 contains the initial and final positions of the obstacle in every task[13]. Note

[10]Note: the `motion_controller` in step 5 is also set to start the arm in a chosen initial configuration to ensure consistent starting conditions. It is then unnecessary to move the arm manually in step 4, if the desired initial configuration is set in the node.

[11]The pose notation used here is composed of the coordinates in x, y, and z followed by the roll, pitch, and yaw euler angles: $\{x, y, z, r, p, y\}$.

[12]In fact, the DMP-planned motion trajectories only define trajectory positions in x, y, and z, and thus do not actively change the orientation of the end-effector during execution.

[13]The final position is not marked in tasks for which the obstacle is static.



(a) On the robot platform

(b) In simulation

Figure 5.10: Kinova Gen3 Arm Configuration

Table 5.5: Initial and goal poses of the end-effector in each task. Cells marked with "-" indicate no change in that value.

		End-effector pose, relative to *world_frame*					
		x (m)	y (m)	z (m)	R (deg)	P (deg)	Y (deg)
Task 1	Initial	0.470	0.004	1.118	129.56	-0.07	90.20
	Goal	0.900	0.000	0.960	-	-	-
Task 2	Initial	0.507	0.004	1.178	109.54	-0.62	91.20
	Goal	0.900	0.050	1.048	-	-	-
Task 3	Initial	0.507	0.004	1.178	109.54	-0.62	91.20
	Goal	1.000	0.050	0.948	-	-	-
Task 4	Initial	0.507	0.004	1.178	92.365	-0.812	92.134
	Goal	-	-	-	-	-	-

that obstacle positions were chosen to ensure an intersection with the arm's trajectory, i.e. an imminent collision, in the nominal motion trajectory case.

5.1.5 Automated Evaluation Testing

An automated testing and evaluation procedure was developed to run batches of trials in simulation and organize the resultant data, compute performance metrics, and generate visualizations for analyses and comparisons. Such an implementation alleviates the necessity of running, monitoring, and processing data from hundreds of task executions manually, and more completely harnesses the advantages of simulation by enabling the execution of experiments with no supervision. This is especially valuable given that the parameter tuning phase would involve waiting for significant periods of time for sufficiently many executions to be run with a given parameter set before drawing conclusions and running subsequent tests. In this section, we briefly describe the products of this automated evaluation, including visualizations that aid in comparing results from different parameter sets or strategies.

Table 5.6: Initial and final positions of the obstacle in each task. Cells marked with "-" indicate no change in that value.

		Obstacle position, relative to *world_frame*		
		x (m)	y (m)	z (m)
Task 1	Initial	0.470	0.004	1.118
	Final	-	-	-
Task 2	Initial	1.050	-0.350	1.050
	Final	0.400	0.250	1.050
Task 3	Initial	0.676	-0.084	0.932
	Final	-	-	-
Task 4	Initial	0.891	0.007	1.178
	Final	-0.109	0.007	1.178

Firstly, the automated evaluation pipeline runs consecutive trials of any task by simply following the experiment procedure outlined in the previous section. Here, the user need only specify the relevant hyper-parameters including the configuration files of each component, the desired initial and final positions of the end-effector and obstacle, the number of trials to be conducted, etc. The pipeline then ensures the seamless execution of the outlined steps for multiple trials and handling any potential failures[14].

The data associated with every trial is saved in a dedicated directory This data includes raw measurements of the positions of the end-effector, robot joints, and objects, as well as the velocities and accelerations of the end-effector. In addition, the quantitative metrics detailed in Table 5.3, can be easily derived from the raw data, are computed and similarly stored. In the case of $N_{collisions}$, we disable Gazebo collision dynamics in order to count instances in which the end-effector's position intersects the obstacle's in all three dimensions at the same time[15]. Finally, the plots described in section 5.1.3 are also generated and saved.

The automated evaluation tools were also developed to enable convenient comparisons of multiple batches of trials. Primarily, the batches to be compared belong to either i) different parameter sets in the tuning and validation phases, or ii) executions with and without SNN-based obstacle avoidance, which constitute the final experiments conducted in the testing phase. The aforementioned run metrics are aggregated for a given batch, and statistics such as the mean, minimum, maximum, percentile scores, and outliers for every quantitative metric's values are computed. These are visualized in hybrid plots such as the one depicted in Figure 5.11, where every column represents a metric, and every entry in the x-axis belongs to a specific batch of trials, whose IDs are listed in the legend. As a binary metric,

[14]More specifically, failures that involve the repeated execution of trials, such as the arm and obstacles not being set to the specified positions before the next run, etc.

[15]Here, we consider the end-effector's position as a point. This is sufficient for the analyses conducted in our experiments, but a better approach would consider the dimensions of the end-effector and the rest of the robot as rigid bodies, but is left for future extensions.

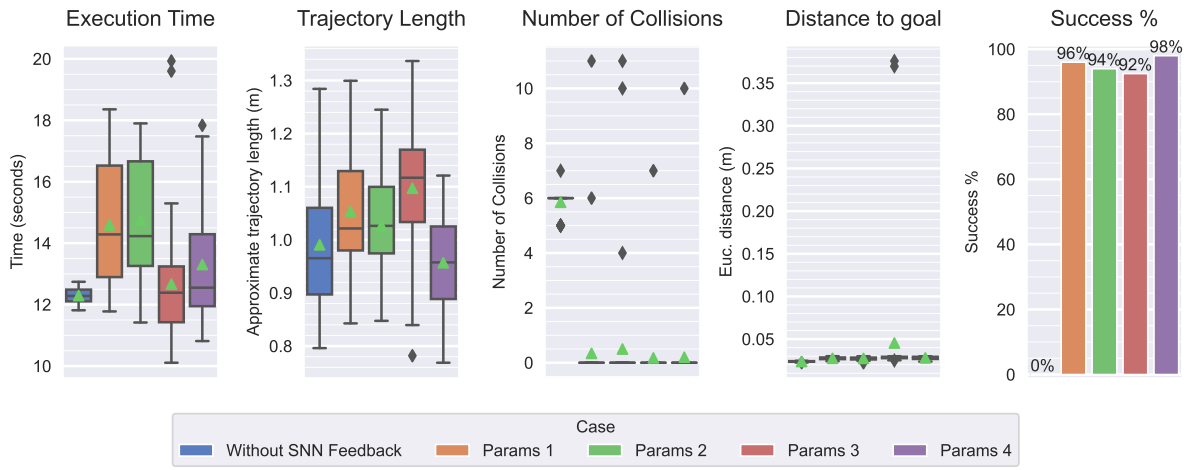
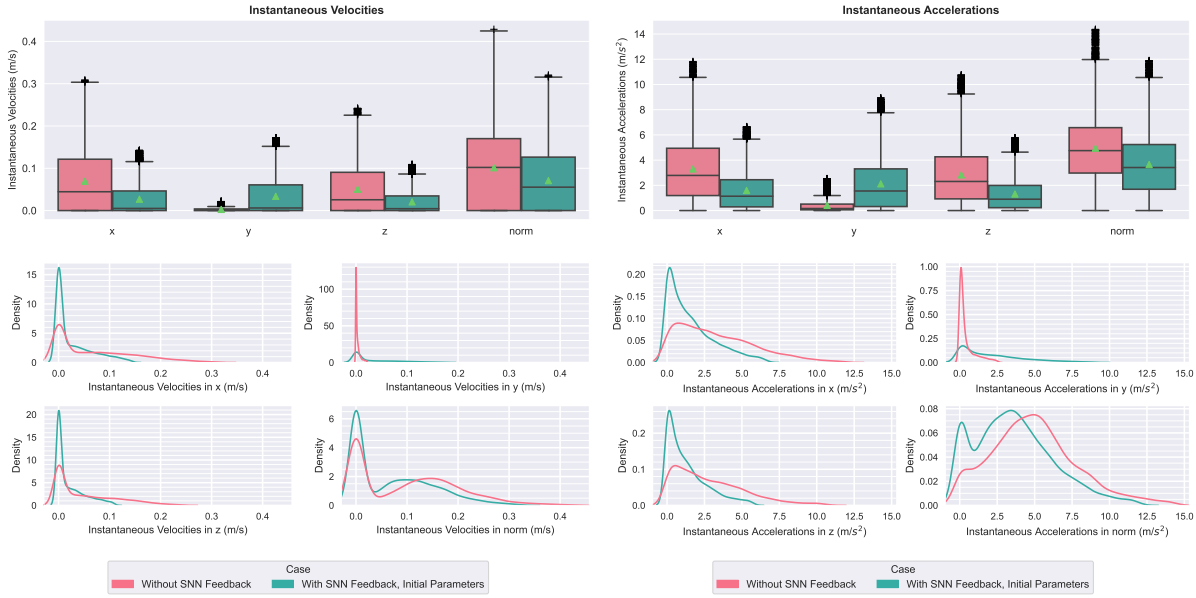


Figure 5.11: A plot comparing the values of the quantitative performance metrics for batches of trials that were run with different parameter sets. This is produced using the automated evaluation tools.



(a) Velocities (b) Accelerations

Figure 5.12: Plots of the distributions of instantaneous velocities and accelerations in each spatial dimension and their combination, measured in two different batches of trials. The bottom curves depict the densities of measured values in each dimension.

execution success is visualized in a bar plot representing the percentage of successful executions in the given batch, while other metrics' distributions are visualized in box plots to highlight the median, mean (green triangles), spread, and outliers. The magnitudes of instantaneous end-effector velocities and accelerations from different batches are visualized in separate plots like the example shown in Figure 5.12[16]. The measurements in each spatial dimension and their average[17] from every run are aggregated for a given batch and depicted in the box plots on the top. The bottom plots illustrate the densities of these values determined using kernel density estimation. Since these measurements are often fairly noisy, particularly at high sampling frequencies, we detect and remove outliers before aggregating the data presented in the plots. The outliers are detected using the standard interquartile range method, where the 25th and 75th percentiles of the data, respectively denoted $Q1$ and $Q3$, are used to determine the interquartile range, $IQR = Q3 - Q1$, with which outliers can be identified under the following criterion:

$$\mathbb{1}_{outlier}(x) = \begin{cases} 0, & \text{if } Q1 - (1.5 \times IQR) < x < Q3 + (1.5 \times IQR) \\ 1, & \text{otherwise} \end{cases} \tag{5.5}$$

Finally, we visualize the three-dimensional trajectories of a given batch of trials in a single plot of the type presented in Figure 5.9, similarly highlighting the success/failure of every execution; an example is shown in Figure 5.13.

The data and plots generated by the automated evaluation tools facilitate rapid testing and evaluation by presenting the results in forms that simplify assessing performance and drawing comparisons. All run data and batch metrics are saved in YAML files for later processing. Appendix B contains samples of the YAML files containing i) the metrics computed for a single trial, and ii) the aggregate statistics of a batch of trials (used for the comparisons described in this section).

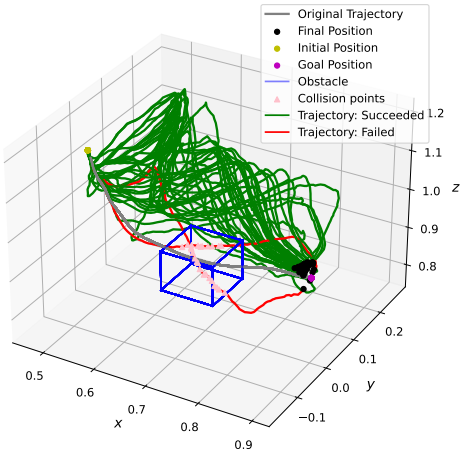


Figure 5.13: 3D trajectory plot for a batch of Task 1 executions (this is similar to the plots in Figure 5.9 but incorporates multiple trials).

5.2 Real Robot Experiments

The real robot experiments are largely similar to the simulation experiments and are conducted on a real Kinova Gen3 arm attached to a robot platform. Following the tuning, validation, and final testing of parameters in the simulation experiments and the selection of a parameter set, these experiments are designed to mimic the task scenarios created in simulation and are performed to evaluate performance in a real setting. In particular, executions on the robot can provide insights on how well the parameters tuned in simulation transfer to the real world and a more

[16]Note that we consider the absolute values of velocities and accelerations, since we are only concerned about comparing their magnitudes, particularly in assessing the qualitative criteria described in section 5.1.3.

[17]The Euclidean norm of the values in $[x, y, z]$.

(a) Start (b) End

Figure 5.14: Task 1 (real robot experiments): reaching for a goal position behind a static obstacle. A nominal end-effector trajectory is illustrated in green.

concrete validation of the proposed approach, in general. These experiments contain fewer variations of task scenarios from a subset of the simulation tasks defined in section 5.1.1, and share the same evaluation metrics/criteria and experimental procedure.

5.2.1 Evaluation Tasks

For these experiments, we consider two of the tasks defined in section 5.1.1[18]:

- **Task 1**: The arm must reach a commanded goal position that lies behind a **static** obstacle, which is positioned between the arm and the goal such that a nominal motion trajectory would lead the end-effector to collide with the obstacle.
- **Task 2**: The arm must reach a commanded goal position as a **dynamic** obstacle enters the field of view and crosses the end-effector's path; the trajectory of the obstacle is set to intersect with a nominal motion trajectory and thus lead to a collision.

The setups of both tasks are shown in figures 5.14 and 5.15, respectively, which depict the start and end of each as well as the nominal end-effector trajectory and the obstacle's trajectory (for task 2).

While their definitions and conditions are essentially equivalent to tasks 1 and 2 of the simulation experiments, the versions used in real experiments have minor differences that bear mentioning. In simulation, the object in task 1 was suspended for simplicity; here, the object is placed on a surface. In task 2, the simulation offers fairly accurate control of the obstacle's motion trajectory (handled by the `gazebo_object_animator` node, as described in section 5.1.4) and thus a high degree of consistency across trials. For the real experiments, the object's trajectory is manually controlled by the experimenter, with steps taken to minimize the variance of trajectories, including guiding markers on the table surface (see Figure 5.15), verification measurements of the object's initial position at the start of every trial, etc.

[18]The only criteria used to select the tasks for the real robot experiments are similarity to the ones tested in simulation, which is important for minimizing external factors when drawing conclusions on transferability from simulation to a real setting, in addition to the inclusion of at least one static and one dynamic case.

(a) Start (b) End

Figure 5.15: Task 2 (real robot experiments): reaching for a goal position in the presence of an interfering dynamic obstacle. The obstacle's trajectory and a nominal end-effector trajectory are illustrated in red and green, respectively.

These steps reduce inconsistencies in trial conditions, but the manual control of the trajectory is likely to introduce some. However, the precise reproducibility of task conditions at this level of detail is deemed unnecessary to achieve the objective of these experiments, and is not as strictly optimized for as in the simulation experiments. One reason is that precise repetitions of conditions are rare if not impossible in the real world in any case. While the simulation experiments were aimed at a systematic evaluation of the approach and its parameters based on statistical evidence, which is facilitated by controlled and highly repeatable trials, the primary objective of real robot experimentation is to transition into real-life conditions and investigate how well the implementation copes with these conditions. From this perspective, the aforementioned subtle variations may in fact be conducive to a thorough evaluation. In addition, provided that this variation in conditions is reasonably small, running multiple trials is expected to lead to an acceptable approximation of average performance, since that is likely to diminish any noise in results that is introduced by imprecisions. The tasks also differ slightly from the simulation versions in the initial and final positions of the robot arm and obstacles (see section 5.2.3 for exact values); the same arguments for the insignificance of these differences in the context of the real robot experiments can be made here[19].

Although the real robot experiment tasks could be varied along the same dimensions defined from which the simulation task scenarios were instantiated (see Table 5.1), we choose a smaller set of representative scenarios here. Both tasks are characterized by a "Background" and "Obstacle Type" variables. The "Obstacle Color" and "Obstacle Speed" variables are not varied during the present experiments. For the first, "Lab Background 1" is situated in an area near a window providing a natural but dim light (see Figure 5.14) while "Lab Background 2" contains a large table and other background objects and has a much brighter artificial lighting (see Figure 5.15), providing some variation in background and lighting conditions. The objects used as obstacles are a wooden block, a metal bar, and a person's hand. A person's hand as an obstacle is a particularly relevant case for human-robot collaborative scenarios and is

[19]However, this is under the assumption that the initial configurations of the arm are still reasonably unrestrictive, as explained in section 5.1.4.

Table 5.7: List of real robot experiment task scenarios

Scenario ID	Set	Scenario Specification
R1	Testing (Real)	Task 1, Lab Background 1, Wooden Block
R2	Testing (Real)	Task 2, Lab Background 2, Hand
R3	Testing (Real)	Task 2, Lab Background 2, Metal Bar
R4	Testing (Real)	Task 2, Lab Background 2, Wooden Block

thus worth testing for[20]. Table 5.7 lists the four scenarios we conduct our real robot experiments in.

5.2.2 Evaluation Metrics and Criteria

The performance in real experiments is evaluated using the same quantitative metrics and qualitative criteria of the simulation experiments, described in section 5.1.3. The only difference is in the $N_{collisions}$ quantitative metric. In simulation, it is possible to disable collision dynamics and use the number of instances in which the end-effector intersects an obstacle model as a useful measure of collision frequency. Since that is not possible in the real experiments, we exclude this metric. Collision instances are instead reflected in the "Success" metric which, as before, is true only if the end-effector never collides with the obstacle and also reaches its designated goal. In addition, note that these instances are determined through observation, instead of the automated collision detection implemented for the simulation experiments. For the definitions of all other metrics and evaluation criteria, in addition to descriptions of plots and other visualizations, refer to section 5.1.3.

5.2.3 Experiment Procedure

Similar to the evaluation metrics and criteria, the procedure followed in conducting the experiments on the real robot is mostly the same as in the simulation experiments. We run the SNN-based obstacle avoidance module with the parameter set that has been tuned, validated, tested, and ultimately selected in simulation experiments. Again, this involves N_{trials} execution trials on the task scenarios (defined in Table 5.7) in two cases:

1. Nominal executions that do not involve the obstacle avoidance module (the benchmark)
2. Executions that incorporate the SNN-based obstacle avoidance module, using the selected parameter set

The real robot experiments consist of $N_{trials} = 30$ trials in both cases for each scenario, followed by the same analysis of the obstacle avoidance behaviour.

The particular steps constituting a single run of the experiments resemble those enumerated in section 5.1.4, with some adjustments pertaining to the real robot platform:

[20]Note that the speed at which the end-effector moves poses negligible safety risks to the person. Nevertheless, the arm is carefully monitored during execution and an emergency stop is available to interrupt the execution at any point.

1. Place the robot platform in the designated starting position.
2. Launch components required to initialize the robot's control functionalities.
3. Start the event camera emulation and SNN simulation components by running the `event_image_streamer` and `snn_simulator` nodes with the appropriate configuration files.
4. Move the arm to the initial configuration (depends on the task).
5. Prepare the obstacle object. In static scenarios, place the obstacle in its designated position. In dynamic scenarios, hold the object in its designated initial position in preparation for its motion.
6. Command the arm to move to the goal position (depends on the task) by running the `motion_controller` node with the appropriate configuration file, and choosing whether to use SNN feedback.
7. If the scenario involves a dynamic obstacle, move the obstacle in its defined trajectory.
8. After task execution, record all relevant data and compute performance metrics.

The components are configured and run with the same configuration files described in section 5.1.4.

The end-effector's designated initial and goal poses as well as the obstacle's positions for each task are marginally different from the simulation task definitions. Tables 5.8 and 5.9 specify the end-effector poses and obstacle positions, respectively, for each task in the real robot experiments. In this case, note that the end-effector's poses are the same in both tasks.

Table 5.8: Initial and goal poses of the end-effector in each task in the real robot experiments. Cells marked with "-" indicate no change in that value.

		End-effector pose, relative to *world_frame*					
		x (m)	y (m)	z (m)	R (deg)	P (deg)	Y (deg)
Task 1	Initial	0.494	-0.025	1.130	129.56	-0.07	90.20
	Goal	1.000	0.000	0.948	-	-	-
Task 2	Initial	0.494	-0.025	1.130	129.56	-0.07	90.20
	Goal	1.000	0.000	0.948	-	-	-

Table 5.9: Initial and final positions of the obstacle in each task in the real robot experiments. Cells marked with "-" indicate no change in that value.

		Obstacle position, relative to *world_frame*		
		x (m)	y (m)	z (m)
Task 1	Initial	0.700	0.000	0.900
	Final	-	-	-
Task 2	Initial	0.790	-0.390	0.950
	Final	0.790	0.000	0.950

Finally, all measurements and data are saved, metrics and comparison statistics from multiple batches are computed, and single-run and batch visualizations are generated in the same manner described in

section 5.1.5.

5.3 Concluding Remarks

This chapter has summarized the methodology we follow for the systematic evaluation of our proposed neuromorphic approach to obstacle avoidance. First, we interweave the iterative development and evaluation of our implementation in the presented tuning → validation → testing procedure in simulations. Simulation testing facilitates the methodical selection of a parameter set and a thorough assessment of performance and obstacle avoidance behaviour through a large number of repeatable trials. By varying task scenarios along several dimensions and conducting experiments on different sets of scenarios, we aim to draw well-founded conclusions and identify the properties and limitations of our approach. The resulting implementation and parameter set are then tested in real robot experiments, particularly by comparing outcomes in imminent collision cases to those of a naïve approach that does not incorporate our obstacle avoidance module. Ideally, these parameters would transfer directly or with minimal adjustments and achieve comparable performance, thus validating not only the SNN-based obstacle avoidance in general, but the transferability of parameters tuned in simulation to the real world.

An integral aspect of our evaluation methodology is the definition and establishment of clear quantitative metrics and qualitative criteria for a practical interpretation of results. We detail the measurements and computations required to derive the metrics and ground a selection of qualitative criteria with which the executed trajectories can be described in terms of quantifiable measures. Given the relative scarcity of similar evaluation tools in the literature, we hope that the metrics and criteria defined here are applied and refined in future work as a step towards an objective and unified methodology for evaluating obstacle avoidance behaviour.

6

Results and Discussion

Following the procedure outlined in chapter 5, we conducted the experiments and analyses designed to evaluate our neuromorphic obstacle avoidance approach and present and discuss their results in this chapter. We first describe the selection of parameters, their evaluations, and subsequent improvements in simulated testing, particularly in terms of the presented performance metrics and criteria. Along with parameter adjustments, we discuss modifications to the implementation prompted by findings from initial testing, such as incorporating safety strategies to conditionally limit arm motions. This is followed by an evaluation of the ultimate performance of the implementation and selected parameter values in the simulation and real robot testing scenarios. The subsequent analysis of the results provides insights on the degree to which our SNN-based strategy succeeds in obstacle avoidance, the effects of the neuromorphic elements of the pipeline, and the strengths and limitations of the approach in general.

We additionally explore the properties of our neuromorphic components through further experimentation. We compare different event emulation methods in terms of output events, spiking responses, and effects on performance. To validate the utility of the SNN component, we implement and test the decoding of obstacle avoidance behaviour from raw events, thus completely excluding the SNN. Furthermore, we investigate the effects of SNN random weight values on performance by repeating scenario trials with different weight initializations. Finally, we substitute our event emulator with a real EC and re-ran experiments on the real robot for a preliminary examination of the feasibility of incorporating a real camera in our pipeline and its effects on obstacle avoidance performance.

This chapter is divided as follows. Section 6.1 details the results of the tuning → validation → testing procedure and simulation experiments. Section 6.2 contains a discussion of the results of real robot experiments, and is followed by a summary of the conclusions we have drawn from both sets of experiments in section 6.3. Our analyses of the results of different emulation methods, the exclusion of the SNN, and varying SNN weights are presented in sections 6.4, 6.5, and 6.6 respectively. In the last section, we discuss the results of preliminary tests with the real camera.

6.1 Simulation Experiments

The parameter tuning and evaluation in the simulation experiments was performed in the manner outlined in section 5.1, but was preceded by first establishing an initial set of parameter values while developing the implementation in a *pre-tuning* phase. Then, we tuned several variants of this initial

parameter set on the tuning set of scenarios, validated their performance on the validation set, feeding back the results to instantiate more parameter sets to tune and validate, and finally selected the best-performing parameter set to run on the testing set. All executions were evaluated based on the quantitative metrics and accompanying visualizations presented in section 5.1.3, which are computed and generated by the automated evaluation tools described in section 5.1.5. The formal simulation experiments conducted by running trials on the testing set yielded the final results with which the performance of the SNN-based obstacle avoidance in simulation was assessed, including the qualitative criteria defined to evaluate motion trajectory characteristics. In the following, we present the results from the pre-tuning, tuning, validation, and testing phases and discuss interesting insights from each. Note that all parameter sets discussed in this section are included in Appendix C.

6.1.1 Initial Parameters (Pre-Tuning Phase)

The pre-tuning phase consisted of iteratively testing the implementation during development and grounding parameter values that lead to acceptable performance. Since the number of tunable parameters is considerable, a significant number of closely monitored execution trials was conducted to search for a promising region in the parameter space. Among others, the main targets of this optimization were motion control parameters, such as the PID gains, distance tolerances, and motion loop frequency, PF parameters, SNN parameters, including the architecture, weight initialization, and SNN dynamics variables, and event emulation options, such as using RGB vs. grayscale images and the event emission threshold (θ). These tests were conducted on a special task scenario that was exclusively used in the pre-tuning phase: a variant of Task 1 containing a "Cracker Box" object excluded from the task distribution defined in section 5.1.1 (Table 5.1). In addition, the arm was positioned in the default configuration of the *ros_kortex* simulation. Otherwise, the scenario follows the formalized task specification. Figure 6.1 shows the setup at the beginning of the task scenario.

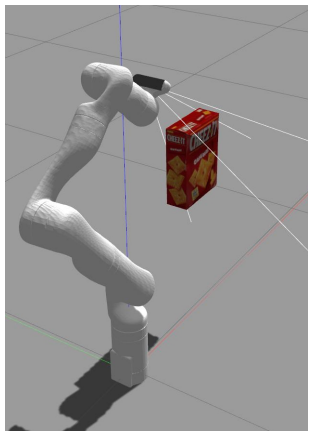


Figure 6.1: Pre-tuning scenario: {Task 1, Empty, Cracker Box}

During this phase, we established stable values for various options and parameters. The event camera emulator was configured to:

- derive events from RGB instead of grayscale images, thus utilizing information from all three color channels (*compute_from_rgb*)
- emit events when the intensity threshold is crossed in all three channels, instead of any one (a multi-channel instead of single-channel strategy: *rgb_multi_channel*)
- record OFF events as well as ON events (*record_off_events*)
- treat OFF events as ON, thus not considering event polarity (as done in other studies; see section 4.1.2) (*register_off_events_as_on*)

Table 6.1: The SNN architecture selected during the pre-tuning phase.

Layer	Type	Kernel Size	Stride Size	Input Size	Output Size
Input	-	-	-	-	120 × 160
Layer 1	LIF (conv)	8 × 8	4 × 4	120 × 160	29 × 39
Layer 2 (Output)	LIF (conv)	4 × 4	4 × 4	29 × 39	13 × 18

For the SNN simulator, the chosen architecture is a two-layer SNN comprising neurons of the type presented in Diehl & Cook (2015), which are essentially leaky-integrate-and-fire (LIF) neurons and are implemented in the *BindsNET* package as *DiehandCookNodes*. The network consists of an input layer which simply propagates the input spike trains and two layers of characteristic spiking neurons (identified with H and Y)[1]. Table 6.1 shows the specifications of each layer, including the convolutional kernel and feature map sizes. (See Appendix B for a representation of the architecture in the SNN configuration YAML file.) We set the parameters that govern neuronal dynamics, such as the spiking threshold (v_{thresh}), reset and resting potentials (v_{reset} and v_{rest}), refractory period (T_{refrac}), and potential decay time constant (τ_v), to values similar to the defaults used in the original paper, which are intended to be within biologically plausible ranges (Diehl & Cook (2015)). Most notably, we increased v_{reset} and v_{rest} (from -65.0 to -62.0) in order to encourage more frequent spiking, which was more suitable in our pipeline. The SNN simulation time (T_{sim}), which controls the time period of a single pass through the network, was set to 20 (ms)[2]. The weight initialization factor, w_c, was set to 7.0 (refer to section 4.1.3).

For the motion controller, the PID gains (K_p, K_d, K_i) and position and final goal reaching tolerances and timeouts were tuned for motions that seemed most stable, safe and successful. A lower bound on the motion loop iteration time of 0.03s, i.e. a maximum frequency of approximately $f_v = 33$ Hz, also contributed to motion stability. Other notable parameters that were tuned include the PF parameters (p_0 and η of the Park method of equation 4.4, a gradient scaling constant, and the ϕ_{max} limit on $\boldsymbol{\phi}$ values), the FST activation threshold factor, t_{act}, that determines how early an output neuron must fire (within T_{sim}) for its contribution to be considered when decoding the SNN response, and the *aggregate_phi* option, which specifies that a history of recent $\boldsymbol{\phi}$ values is averaged to alleviate observed noise.

Appendix C contains the selected values of all the parameters (some of which were not mentioned here). For descriptions of each parameter, refer to tables 4.1, 4.2, and 4.3 in section 4.2.1.

[1]Note that the network thus effectively contains three layers, but we only consider layers of active spiking neurons, i.e. those that emit spikes according to tracked membrane potential traces, when categorizing the architecture. Therefore, we do not count the input layer, which simply transmits input spikes to the main layers.

[2]This means that membrane potentials are updated, spikes are propagated, etc. in all neurons for input spike trains that span a period of 20ms before reading off the spikes at the output layer. The length of spike trains is enforced during the Poisson spike encoding of input event data described in section 4.1.3. This creates a brief time window in which the *snn_simulator* output is halted until this outcome is determined. In future work, we aim to explore this parameter further, particularly by setting its value to the minimum time increment (currently 1 ms) such that the SNN runs effectively in real-time, and outputs are read after a maximum of a single spike is propagated per neuron.

Pre-Tuning Test Results

The pre-tuning parameter set was formally evaluated by running $N_{trials} = 30$ trials and comparing to results conducted without the obstacle avoidance module. We refer to the latter control trials as executions "without SNN feedback"[3], and use their results as a benchmark. Figure 6.2 depicts the performance in either case based on the quantitative metrics, Figure 6.3 shows the 3D trajectories executed in both batches, and Figure 6.4 depicts the distributions of applied velocities and accelerations.

The trajectories executed in the control case all fail by colliding with the obstacle, as is clear from the consistent red trajectories in Figure 6.3a. This is expected, since no obstacle avoidance trajectory adaptations are performed. On the other hand, utilizing the SNN-based obstacle avoidance leads to trajectories that are adapted to avoid the obstacle while moving towards the goal and which often succeed (Figure 6.3b). In particular, these executions were successful 80% of the time (as opposed to 0%) as depicted on Figure 6.2. The obstacle avoiding trajectories tend to have higher execution times and trajectory lengths, approximately by factors of 2 and 1.5, respectively. In addition, their values have higher variances about the mean; this is a reflection of some differences in the executed trajectories across trials, which can also be observed from their shapes on Figure 6.3b. Nevertheless, the obstacle avoidance module reduces the average number of collisions from approximately 7.8 to 0.6 (a factor of 14), though a few outliers indicate that failed executions occasionally lead to high counts of collisions. A significant fraction of failures occur due to executions ending before the end-effector has reached the goal, i.e. with a high d_G that exceeds $\delta_{\mathbf{g}}$ (the goal reaching tolerance). This is evident from the slightly larger mean and the significant number of outliers in the distance-to-goal results from the "With SNN Feedback" run in Figure 6.2, as well as a number of red trajectories in Figure 6.3b whose final positions are far from the goal[4]. Clearly, the chosen parameters lead to occasional instances where the applied velocities move the arm into singular positions that complicate returning back to the intended path.

From Figure 6.4, we can observe that velocities and accelerations applied by the obstacle avoidance module are not significantly different from the baseline values recorded in nominal executions. The box plots show that the overall speed is actually lower with obstacle avoidance, particularly in the x and z dimensions. This is due to trajectory adaptations naturally slowing down the overall progression of the end-effector, since adapting online to the reach adjusted DMP positions requires more corrective control actions from the motion controller and thus more time. In the exceptional case of the y dimension, obstacle-avoiding velocities and accelerations are higher, because the nominal trajectories move very little in that direction in this task specification, while adapted trajectories often do in the course of avoiding the obstacle. These observations are confirmed by the density plots at the bottom halves of each figure.

Having identified potential problems with this parameter set (particularly failures to reach the goal), we transitioned to the tuning phase in which we refine derivatives of the pre-tuning parameter set.

[3]This is synonymous with "without the obstacle avoidance module/pipeline".

[4]Note that trials are bounded by a goal reaching timeout parameter, which is set to 60 seconds by default. If a given trial does not end by reaching the goal before the timeout, it is aborted and the position of the end-effector at that point in time is recorded as its final position.

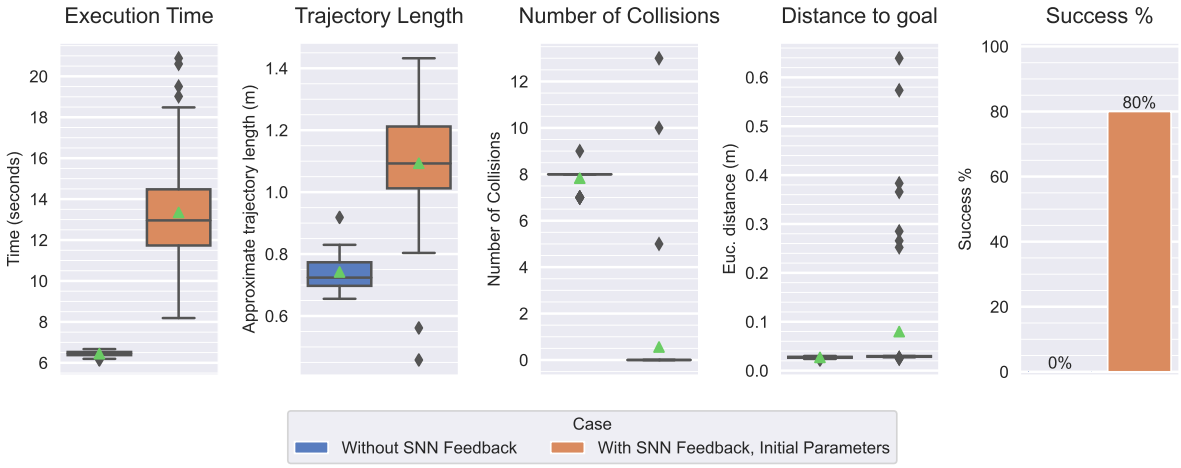
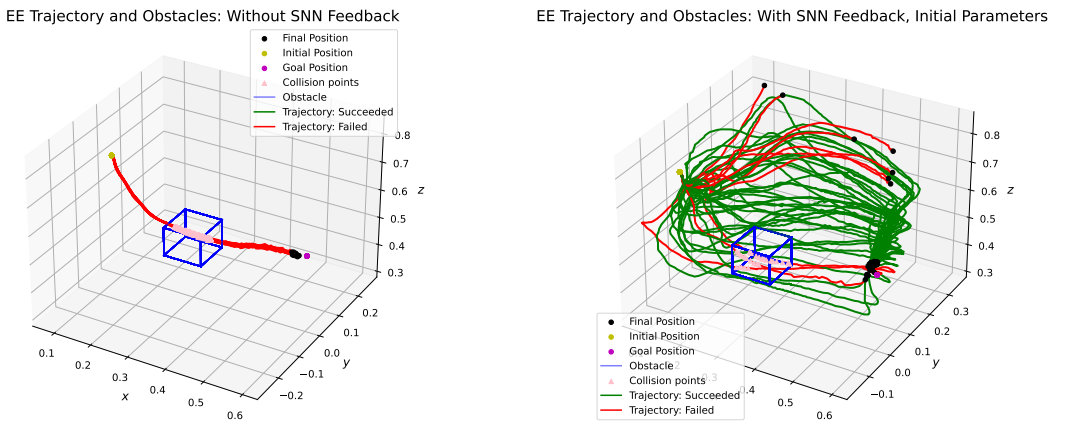


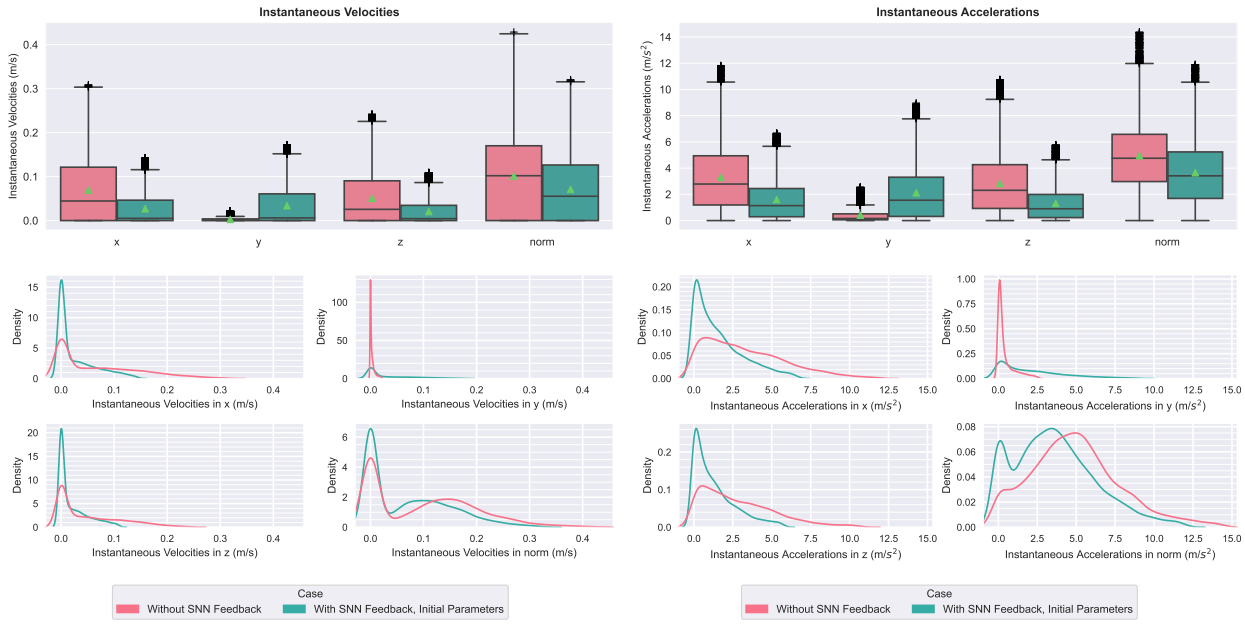
Figure 6.2: Quantitative metric results in pre-tuning trials.



(a) Without SNN feedback

(b) With SNN feedback: Pre-tuning Parameter Set

Figure 6.3: Trajectories executed in pre-tuning trials.



(a) Velocities

(b) Accelerations

Figure 6.4: Distributions of instantaneous velocities and accelerations in each spatial dimension, measured during pre-tuning trials.

6.1.2 Tuning Results

In all subsequent scenarios, the arm in placed in the experimental configuration described in section 5.1.4 and shown on Figure 5.10b.

Given the main conclusions from the pre-tuning tests, we derived parameter sets from the pre-tuning set that were expected to improve its results. Primarily, we aimed to address the problem of the arm reaching singular configurations, which not only leads to failures in reaching the goal, but also to potentially unsafe executions. We devised a safety strategy that discourages excessive motion away from the pre-planned (nominal) motion trajectory in order to minimize similar occurrences. This is termed “Safety Strategy 1”, and is detailed in the following.

Safety Strategy 1: If the distance of the current end-effector position from the nearest point on the pre-planned trajectory exceeds a safety threshold:

$$||\mathbf{y}(t) - \hat{\mathbf{y}}_{ref}||_2 < \delta_{safety,1} \tag{6.1}$$

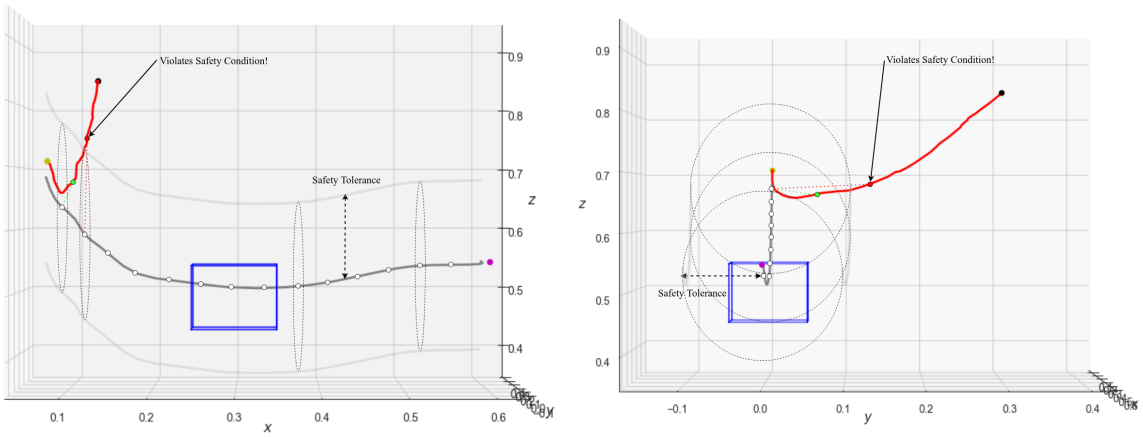
and the current position is further away from the trajectory point than the last recorded position (i.e. the end-effector is currently moving further away from the nominal trajectory):

$$||\mathbf{y}(t) - \mathbf{y}_{ref}||_2 > ||\mathbf{y}(t-1) - \mathbf{y}_{ref}||_2 \tag{6.2}$$

then slow down motion by reducing the current commanded velocity and the $\boldsymbol{\phi}$ values that define the next commanded acceleration values:

$$\mathbf{v}(t) = \gamma_{v,1}\mathbf{v}(t) \tag{6.3}$$

$$\boldsymbol{\phi}(t+1) = \gamma_{a,1}\boldsymbol{\phi}(t) \tag{6.4}$$



(a) X-Z

(b) Y-Z

Figure 6.5: An illustration of the bounds introduced by safety strategy 1, overlaid on a plot of a sample failed execution from the pre-tuning batch. White points are nominal trajectory positions, the green and red points represent a safe position and a position that violates the safety condition. Ellipses represent the maximum distances around nominal trajectory positions, forming the safety boundary defined by the tolerance value.

Figure 6.5 illustrates the safety boundaries that would be established by applying the safety strategy to a sample failed execution from two perspectives. The distance tolerance, $\delta_{safety,1}$, effectively creates an elliptical boundary around the nominal trajectory which encompasses any position that does not violate 6.1 (as depicted with ellipsoids centered on trajectory positions and grey boundaries matching the trajectory shape). Any position in the adapted trajectory that falls outside the boundary and is further from the last recorded position (the second condition, 6.2) is subject to a reduction in velocities and

accelerations (6.3 and 6.4). The second condition ensures that motions that move the end-effector back to the bounded region are not penalized, i.e. slowed down, as well. The distance of an end-effector position to the nominal trajectory is approximated by the Euclidean distance to the nearest known position (the white points in the figure[5]) on trajectory $\hat{\mathbf{Y}}$, $\hat{\mathbf{y}}_{ref}$[6]:

$$\hat{\mathbf{y}}_{ref} = argmin_{\hat{\mathbf{y}}_j \in \hat{\mathbf{Y}}} ||\mathbf{y}(t) - \hat{\mathbf{y}}_j||_2 \tag{6.5}$$

For the fundamentally different task 4, we defined a similar version of the rule: "Safety Strategy 2". Task 4 requires the robot to maintain the end-effector's position, move away to avoid obstacles, and return to that position; therefore, it does not involve a reference trajectory. Instead, we simply penalize motions that deviate a certain distance from the set-point position (initial/goal position, $\mathbf{y}(\mathbf{0})$) in the same manner:

Safety Strategy 2: If the distance of the current end-effector position from the set-point position exceeds a safety threshold:

$$||\mathbf{y}(t) - \mathbf{y}(0)||_2 < \delta_{safety,2} \tag{6.6}$$

and the current position is further away from the set-point position than the last recorded position (i.e. the end-effector is currently moving further away from the set-point):

$$||\mathbf{y}(t) - \mathbf{y}(0)||_2 > ||\mathbf{y}(t-1) - \mathbf{y}(0)||_2 \tag{6.7}$$

then slow down motion by reducing the current commanded velocity and the $\boldsymbol{\phi}$ values that define the next commanded acceleration values:

$$\mathbf{v}(t) = \gamma_{v,2}\mathbf{v}(t) \tag{6.8}$$

$$\boldsymbol{\phi}(t+1) = \gamma_{a,2}\boldsymbol{\phi}(t) \tag{6.9}$$

Both safety strategies are integrated in the parameter sets and tested as described in the following.

Parameter Sets 1-9

By running preliminary tests on the tuning scenarios, we tuned multiple parameter sets (derived from the initial set) that essentially incorporate safety strategy 1 but with different parameterizations. These sets are specified in the following with emphasis on their distinguishing characteristics:

- **Parameter set 1**: $\delta_{safety,1} = 0.4$, $\gamma_{v,1} = 0.6$, $\gamma_{a,1} = 0.4$ (safety strategy 1)
- **Parameter set 2**: $\delta_{safety,1} = 0.4$, $\gamma_{v,1} = 0.6$, $\gamma_{a,1} = 0.6$ (safety strategy 1)
- **Parameter set 3**: $\delta_{safety,1} = 0.2$, $\gamma_{v,1} = 0.8$, $\gamma_{a,1} = 0.6$ (safety strategy 1)
- **Parameter set 4**: $\delta_{safety,1} = 0.2$, $\gamma_{v,1} = 0.8$, $\gamma_{a,1} = 0.4$ (safety strategy 1)

[5] Note that the spacing between positions on the nominal trajectory shown on the figure does not accurately reflect the actual resolution at which positions are spaced along the trajectory and only serves to aid in the illustration.

[6] $\hat{\mathbf{y}}$ refers to *a* position on the original, pre-planned trajectory, while the added subscript in $\hat{\mathbf{y}}_{ref}$ is used to identify a reference position on that trajectory that is nearest to the end-effector position, for the purposes of this discussion.

During these tests, we noticed frequent failures in tuning scenario 3, an instance of task 3, when unfortunate motions would lead the end-effector to move towards the edge of or sometimes under the table, leading to terminal collisions. In order to address this and similar task-specific constraints, we implemented optional positional limits on the effective "workspace" of the robot that can be specified to prevent moving towards certain regions, as an additional safety measure. In this case, we influence the arm's motion at the level of planned end-effector positions; if a position produced by the DMP violates defined limits, it is adjusted to keep the end-effector within bounds. More specifically, if the position falls outside a specified range, if any, it is clipped to the maximum value of that range

$$\mathbf{y}_i = min(\delta^+_{pos,i}, max(\delta^-_{pos,i}, \mathbf{y}_i(t))), \forall i \in \{x, y, z\} \tag{6.10}$$

Here, $\delta^-_{pos,i}$ and $\delta^+_{pos,i}$ represent the lower and upper positional limits along dimension i; by default, these are set to $-\infty$ and ∞, respectively. Generally, these limits could be set according to task and environment conditions. We include a parameter set that incorporates a lower positional limit in z which we expect could lead to better performance in scenario 3 by preventing motions that cross the table edge:

- **Parameter set 5**: $\delta_{safety,1} = 0.2$, $\gamma_{v,1} = 0.8$, $\gamma_{a,1} = 0.4$ (safety strategy 1); $\delta^-_{pos,z} = 0.87$

In subsequent sets, we introduced the binary erosion filter for the event data (described in section 4.1.2), which mainly necessitated adjusting the SNN initial weight factor, w_c. The following sets represent different parameterizations based on parameter set 5 that incorporate this filter, mainly varying in the SNN initial weight factor, w_c, and the size of the filter's structuring element, s_{BE}, which controls the minimum size a matrix of 1's in a binary image must be to not be filtered out:

- **Parameter set 6**: $s_{BE} = 3$ (binary erosion), $w_c = 7.0$
- **Parameter set 7**: $s_{BE} = 3$ (binary erosion), $w_c = 15.0$
- **Parameter set 8**: $s_{BE} = 5$ (binary erosion), $w_c = 20.0$

An additional parameter set was created to address limitations observed with the last three, particularly in scenario 2 (see in results below), and to incorporate "Safety Strategy 2". Among these changes were re-tunings of the position reaching tolerance, $\delta_{\mathbf{y}}$, ϕ_{max} of the PF, and the previously infinite $\boldsymbol{\phi}$ history horizon, n_ϕ:

- **Parameter set 9**: $s_{BE} = 4$ (binary erosion), $w_c = 20.0$, $\delta_{\mathbf{y}} = 0.01$, $\phi_{max} = 5000$, $n_\phi = 2$, $\delta_{safety,2} = 0.2$, $\gamma_{v,2} = 0.8$, $\gamma_{a,2} = 0.8$ (safety strategy 2)

(Note that this parameter set was based on set 8.)

Furthermore, three more parameter sets were tuned to refine the significant changes introduced in set 9 further:

- **Parameter set 10**: $s_{BE} = 5$ (binary erosion), $w_c = 20.0$, $\delta_{\mathbf{y}} = 0.01$, $\delta_{obs} = 0.15$, $\phi_{max} = 4000$, $n_\phi = 12$, $\gamma_{a,2} = 0.4$ (safety strategy 2)

- **Parameter set 11**: $s_{BE} = 5$ (binary erosion), $w_c = 25.0$, $\delta_{\mathbf{y}} = 0.01$, $\delta_{obs} = 0.15$, $\phi_{max} = 4000$, $n_\phi = 12$, $\gamma_{a,2} = 0.4$ (safety strategy 2)
- **Parameter set 12**: $s_{BE} = 5$ (binary erosion), $w_c = 20.0$, $\delta_{\mathbf{y}} = 0.01$, $\delta_{obs} = 0.15$, $\phi_{max} = 4000$, $n_\phi = 12$, $\gamma_{a,2} = 0.6$ (safety strategy 2)

δ_{obs} represents the minimum distance the end-effector must be to the goal for obstacle avoidance behaviours to remain active, the purpose of which is to explicitly prevent avoiding a goal object as an obstacle, such as the can on the table in scenario 3. Other re-parameterizations included increasing n_ϕ to reduce noise in $\boldsymbol{\phi}$ values by increasing the history horizon, testing different combinations of s_{BE} and w_c, and adjusting $\gamma_{a,2}$ to experiment with different speed reduction magnitudes in cases of violated safety constraints. The last sets were in fact tuned with feedback from the validation results (presented in the next sub-section), but are introduced here for convenience.

Refer to Appendix C for a tabular representation of these parameter sets, particularly highlighting the differences between each.

As in the pre-tuning phase, we ran batches of trials without SNN feedback (control case) and with SNN feedback, the latter this time with each of the twelve parameter sets, on each of the three tuning task scenarios (1, 2, and 3). The control case batches consisted of $N_{trials} = 30$ trials, while the parameter set trials consisted of $N_{trials} = 40$ each[7]. The tuning phase trials amount to a total of 1530.

Scenario 1 Results

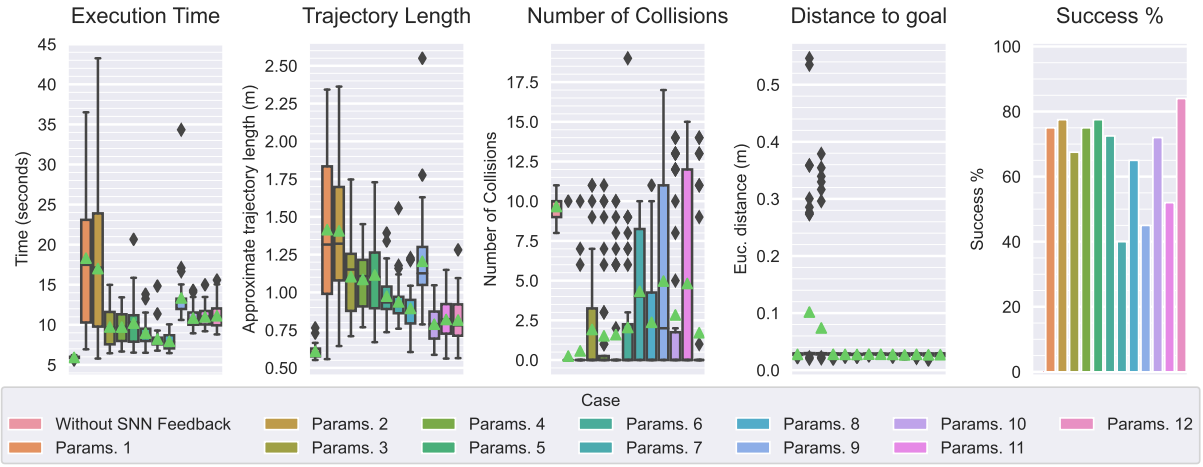


Figure 6.6: Quantitative metric results for tuning scenario 1: without SNN feedback and parameter sets 1-12. Note that the boxes/bars in each plot are arranged in the order: "Without SNN Feedback" and parameter sets 1-12.

[7]We ran less trials of the case not involving obstacle avoidance, given that we expect very little, if any, differences between trials, since the arm simply executes the pre-planned trajectory with no adaptations.

The results for scenario 1, plotted on Figure 6.6, indicate that the first five parameter sets perform acceptably, achieving success rates within the range of 68-78%. Evidently, the safety strategy was successful in reducing instances of terminal configuration, but more so for parameter sets 3-5 than 1 and 2. This can be observed from the distribution of d_G values, which is consistently at the minimum value for sets 3-5, and from the trajectories plotted in Figure 6.7, in which the final positions are near the goal position for all executions of these sets. Indeed, the trajectories for sets 1 and 2 appear more erratic and winding, which is additionally reflected on their higher execution times and trajectory lengths.

Contrarily, the results from parameters 6-9 were worse on average, as can be observed through their success rates (the middle four bars on the metrics plot). This seems to be most attributable to more frequent collisions, likely due to narrower motions trajectories, when comparing the trajectories (shown in Figure 6.8) to those of the first five parameter sets. The greater suppression of motions is an effect of the added binary erosion filter; although it is expected to stabilize motions in noisier backgrounds (such as those in the validation set), the filter does reduce the density of emitted events which may have a detrimental effect in simpler environments such as in scenario 1. The results thus indicate that event filtering could be improved, which was attempted in sets 10-12.

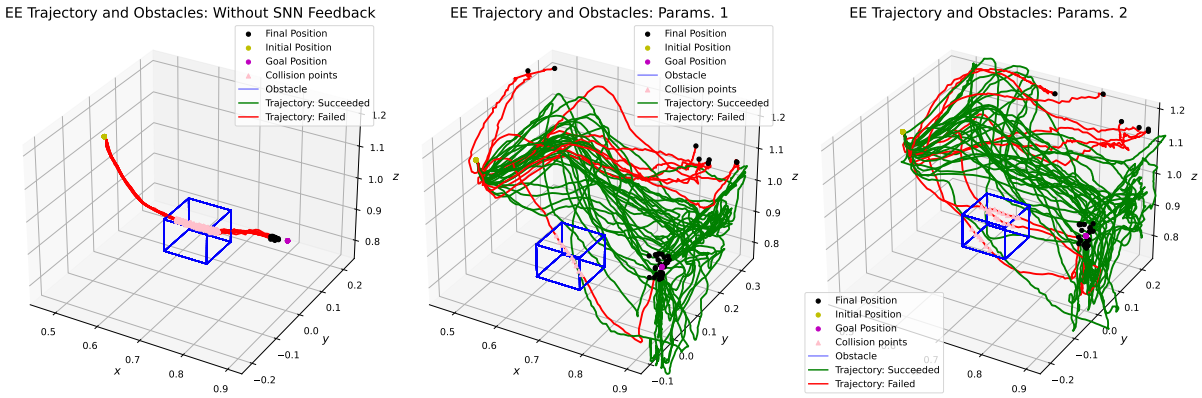
The results show that the further tuning of set 9 to create sets 10-12 has led to better performance. On average, sets 10-12 produce less erratic trajectories than all previous parameters (see Figure 6.8), which is evident from the relatively low mean and variances of trajectory lengths. In addition, parameter sets 10 and 12 perform relatively well in avoiding collisions while consistently reaching the goal, with the latter succeeding at a rate of 84%. The exception, set 11, does not perform as well, seemingly due to multiple cases in which the end-effector does not maintain motion away from the obstacle long enough to completely clear it.

Scenario 2 Results

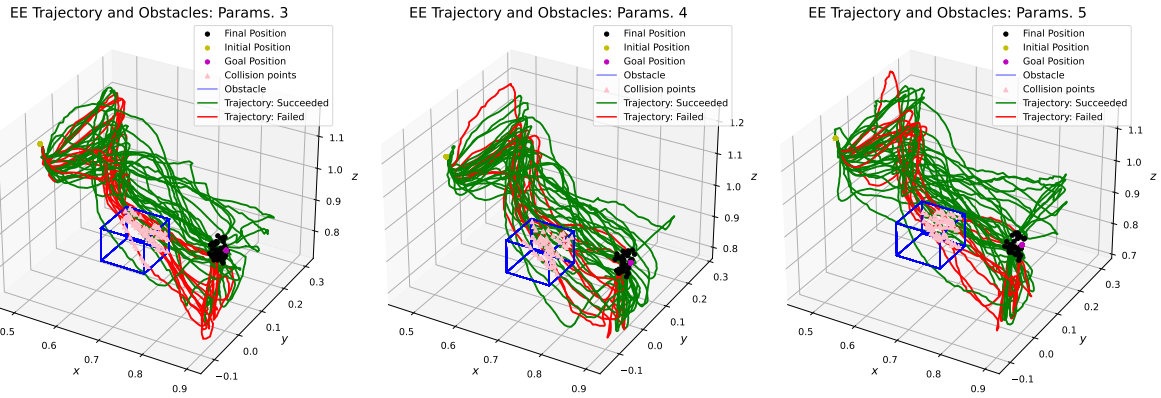
Results from this scenario indicate that most parameterizations fail to achieve acceptable performance in task 4. Among the first nine parameter, the most successful, set 8, has a success rate of 40%, while the rest fall under 17%. The primary cause is large number of collisions, as evident from Figure 6.9, caused by the arm not moving sufficiently away from the obstacle before colliding; this indicated that the inputs were not sufficiently salient or did not induce a strong enough response from the obstacle avoidance component. Essentially, this points to a trade-off in parameter tuning: increasing sensitivity to inputs at the perceptual, motion, or intermediate levels in the pipeline may lead to excessive or oscillatory motions, approaching singular configurations, and more dangerous motions, while decreasing sensitivity may risk not reacting fast enough to avoid a collision[8]. Examples of parameters that can control this sensitivity include:

- Perceptual: event emission threshold, θ (lower values increase sensitivity); binary erosion structure size, s_{BE} (lower values increase sensitivity)

[8]Here, we use the term sensitivity to qualify the extent to which the arm moves in reaction to motion observed in the source image, which is dictated by the aggregate behaviour of multiple components in the pipeline.

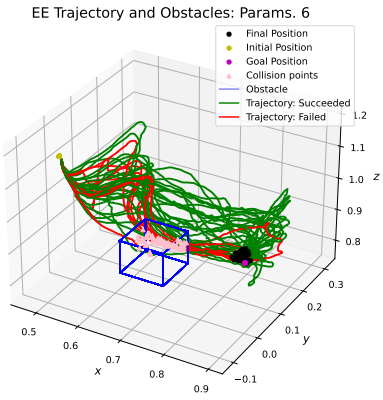


(a) Without SNN feedback (b) Parameter Set 1 (c) Parameter Set 2

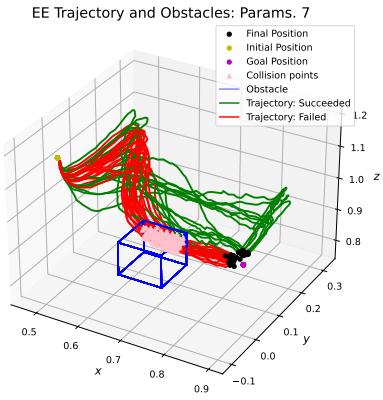


(d) Parameter Set 3 (e) Parameter Set 4 (f) Parameter Set 5

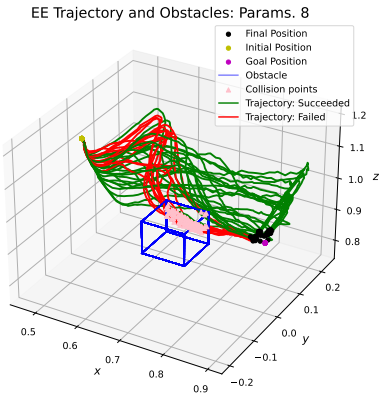
Figure 6.7: Trajectories executed in tuning scenario 1: without SNN feedback and parameter sets 1-5.



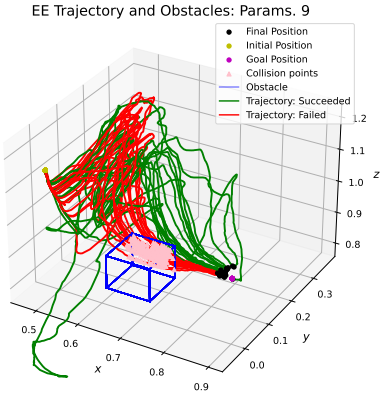
(a) Parameter Set 6



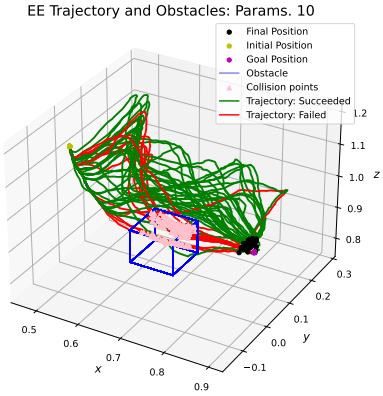
(b) Parameter Set 7



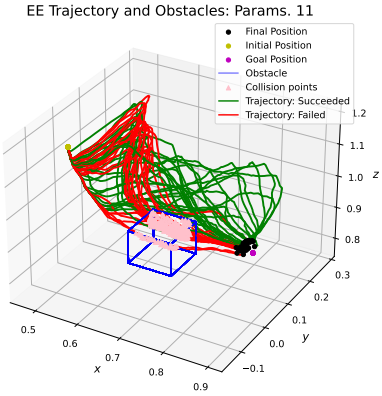
(c) Parameter Set 8



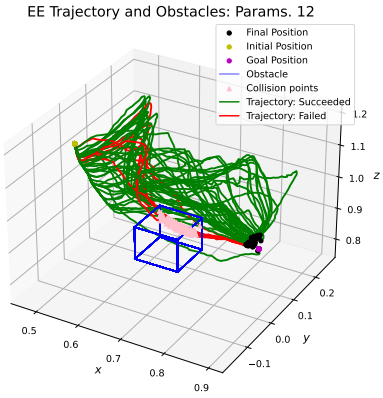
(d) Parameter Set 9



(e) Parameter Set 10



(f) Parameter Set 11



(g) Parameter Set 12

Figure 6.8: Trajectories executed in tuning scenario 1: parameter sets 6-12.

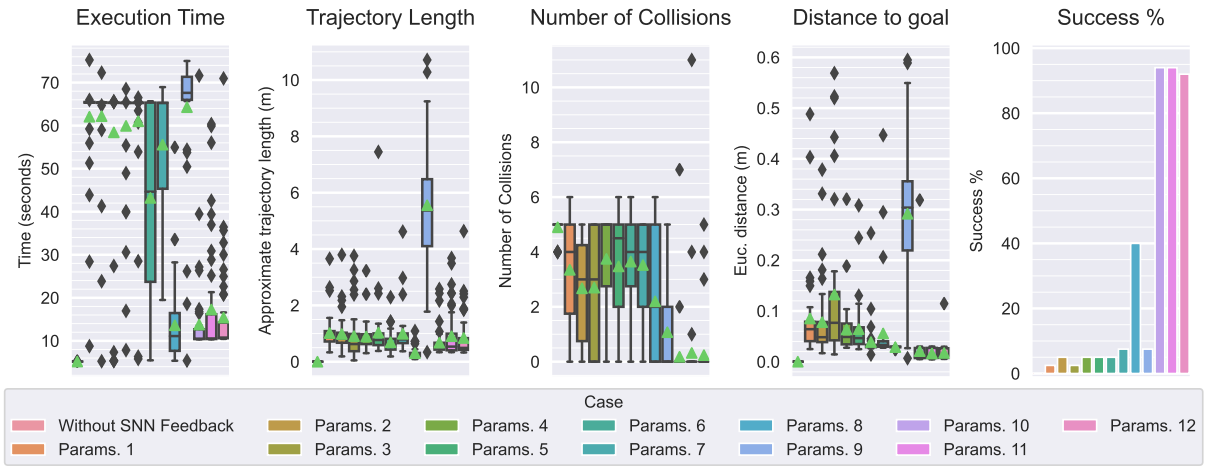


Figure 6.9: Quantitative metric results for tuning scenario 2: without SNN feedback and parameter sets 1-12.

- SNN: SNN weight factor, w_c (higher values increase sensitivity); SNN spiking threshold, v_{thresh} (lower values increase sensitivity)
- Motion: acceleration history horizon, n_ϕ (lower values increase sensitivity); upper bound on acceleration, ϕ_{max} (higher values increase sensitivity)

Sets 10-12 were created in attempts to further tune these parameters to address this deficiency.

As the plot indicates, the parameter values in sets 10-12 succeed in achieving significantly better performance in scenario 2. This validates the hypothesized effects of each of the aforementioned parameters, which we were able to tune in order to reach a satisfactory level of reactivity to accomplish the task. Nevertheless, it is clear that the dynamic component of this task presents an added challenge.

Scenario 3 Results

Figure 6.10 shows the metrics plot for all parameter sets in the third tuning scenario. Here, the success rate for parameter sets 1 and 2 is fairly low at ∼40%, but this improves with the subsequent re-tunings of the safety strategy parameters in sets 3-5, which achieve 64-73%. The initial introduction of binary erosion filter leads to a drop in success with set 6 (∼55%); however, this improves when the filter parameters are tuned further in sets 7 and 8, which is successful 80% of the time. Parameter set 9, on the other hand, performs much worse due to excessive motions leading to unstable trajectories that often fail in reaching the goal (and thus have high d_G values). This is likely in part due to the smaller value of n_ϕ, for which only the past two $\boldsymbol{\phi}$ values are aggregated instead of an infinite time horizon; this increases susceptibility to variances in consecutive accelerations. In general, most failures seem to be due to sensitive parameterizations, since the resulting unstable motions are particularly unsuitable in a scenario that involves other objects in the vicinity (i.e. the table and can).

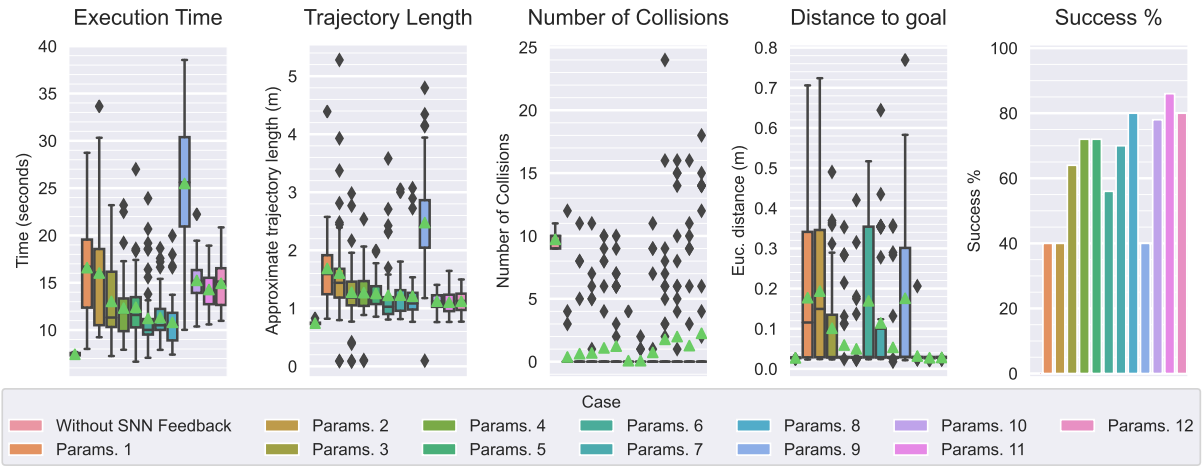


Figure 6.10: Quantitative metric results for tuning scenario 3: without SNN feedback and parameter sets 1-12.

The final three parameter sets seem to address the problem observed with set 9, with sets 11 and 12 achieving 86% and 80% success, respectively. In particular, the resultant trajectories appear much more stable while avoiding the obstacle to successfully reach the goal (see Figure D.2 in Appendix D, which contains all trajectories executed in scenario 3). This indicates that the higher value of n_ϕ within these sets is better, and that the tuning for scenario 2 has produced a better-performing combination of parameter value for this scenario as well.

6.1.3 Validation Results

Following an analysis of the results from the tuning phase, we selected a subset of the twelve parameter sets on the basis of best average performance across the tuning scenarios and most promising trajectory properties: 5, 8, 10, and 12, for the validation phase tests. This phase involves the same procedure of repeated trials, this time conducted in scenarios 4-11.

In preliminary tests, we found that the performance of all parameter sets was not satisfactory for a special task 4 scenario in which the dynamic obstacle moved at a high speed: scenario 8. In order to address this, we instantiated two more parameter sets before running the full validation phase tests, with the aim of studying whether a quicker response that handles the high-speed obstacle could be achieved. We explored the possibilities of faster real-time performance by switching to a single-layer SNN, increasing the upper limit on motion loop frequency, f_v, and adjusting various other parameters to adapt accordingly. The following includes some of the major changes that characterize the two sets, which were based on set 12, apart from the different SNN architecture:

- **Parameter set 13**: $s_{BE} = 5$ (binary erosion), $w_c = 2000.0$, $v_{thresh} = -58.0$, $T_{refrac} = 0$, $T_{sim} = 3$, $\tau_v = 1000$, $\phi_{max} = 8000$, $\eta = 600$, $C_\delta = 40000$, $n_\phi = 12$, $t_{act} = 3$, $f_v = 100$

- **Parameter set 14**: $s_{BE} = 5$ (binary erosion), $w_c = 2000.0$, $v_{thresh} = -58.0$, $T_{refrac} = 0$, $T_{sim} = 3$, $\tau_v = 1000$, $\phi_{max} = 6000$, $\eta = 600$, $C_\delta = 10000$, $n_\phi = 12$, $t_{act} = 3$, $f_v = 100$

Table 6.2 summarizes the single-layer architecture, whose YAML specification is also included in Appendix B. These parameter sets can be found in tabular form in Appendix C.

We ran batches of $N_{trials} = 30$ trials without SNN feedback (control case) and $N_{trials} = 40$ with SNN feedback, with each of the selected parameter sets, on each of the eight validation set scenarios (4-11): a total of 2160 trials. In the following, we briefly report on the results, of the validation set trials, focussing on the most notable insights. The quantitative results from each scenario are plotted in Figures 6.11–6.18.

From among the first three scenarios (all of task 1), the selected parameters generally perform worse in the first two (4 and 5) and relatively better in the third (6) compared to the task 1 scenario included in the tuning set. While not immediately evident by observing the executed trajectories, the obstacle avoidance component seems to have more difficulties in the "Office" environment, in which scenarios 4 and 5 are situated, than "Store" or "Empty". The two most likely causes are the relatively lower illumination, which decreases contrasts between the background and the obstacle thus leading to less events being generated and in turn more latency in or lower magnitudes of avoidance velocities, and the relative clutter in the background which, despite event filtering, may produce more background events that saturate the overall response of the SNN, instead of a response localized around the obstacle points in the feature space. The highest success rate achieved in this task is with set 12 in scenario 6: 94%. Sets 13 and 14, while producing relatively more erratic trajectories, were still as likely to succeed as the other sets.

The results from the task 4 scenarios (7-9) varied significantly. With the object traveling at medium speed (scenario 7), most parameter sets perform well; in particular, set 12 was successful in every trial. However, the high-speed obstacle of scenario 8 (traveling about twice as fast) revealed a significant limitation in the selected sets (5, 8, 10, and 12), which exhibited very low levels of success that did not exceed 10%. In these cases, the obstacle avoidance component would often react to the obstacle, which was visible for a shorter amount of time than the medium-speed obstacle, but the eventual reaction would not be effective enough to clear the obstacle before a collision occurred. This was a product of less events being sensed, the resulting lower activation within the SNN, and the limited speed of the arm. The aforementioned tuning of sets 13 and 14 specifically to address this yielded more positive results, with 33% and 23% more successes than the previous best performance, respectively. Nevertheless, this was still deemed unsatisfactory, and was facilitated by stronger responses and higher instantaneous accelerations from the "faster" parameter sets, leading to less predictable and seemingly less safe trajectories. In

Table 6.2: The single-layer SNN architecture deployed for parameter sets 13 and 14. A 2×2 horizontal and vertical padding is also applied to the input image.

Layer	Type	Kernel Size	Stride Size	Input Size	Output Size
Input	-	-	-	-	120×160
Layer 1 (Output)	LIF (conv)	8×8	4×4	120×160	30×40

addition, these improvements are not tangible in the medium-speed case nor the low-speed case of scenario 9, in which the original parameters outperform 13 and 14 once again (except for set 5), though overall performance appears to be worse than in scenario 6. As hypothesized in section 5.1.1, slower obstacles may produce less salient event activity and thus a less effective response, which may explain the worse results. In this case, set 10 achieves the highest success at ∼93%, followed by set 12 at ∼84%.

Performance in scenarios 10 and 11 was generally similar to the tuning set task 3 scenario, with success rates spanning an average 60%-80%, bar a couple of outliers. The final parameter sets from the initial group each perform best at one of the scenarios: set 12 at scenario 10, and set 11 at scenario 11, out-performing sets 13 and 14.

As expected, all selected parameter sets perform at least better than having no active obstacle avoidance, and reasonably well in most scenarios. An exception is the high-speed task 4 scenario, where the failures we observe may be due to a fundamental limit on how fast we can command the arm to move before approaching dangerous speeds. As a case in point, parameter sets 13 and 14 perform better in this scenario, but exhibit significantly higher accelerations that necessitate the sudden reactions required to address this limitation (though their mean instantaneous velocities are within the collective average). This is characteristic of unsafe trajectories, as described in our discussion of the safety qualitative criterion in section 5.1.3, and presents an undesirable compromise. Instead, it may be reasonable to acknowledge a fundamental inability to reliably react to and avoid obstacles whose speeds exceed a certain threshold in the current design. Such insights validate the benefits of evaluating on a wide distribution of task scenarios, as described in section 5.1.1. Finally, the results of set 12 (and, to some extent, set 10) in comparison to the initial sets indicate that the tuning procedure was fairly successful in optimizing for more successful obstacle avoidance, on average.

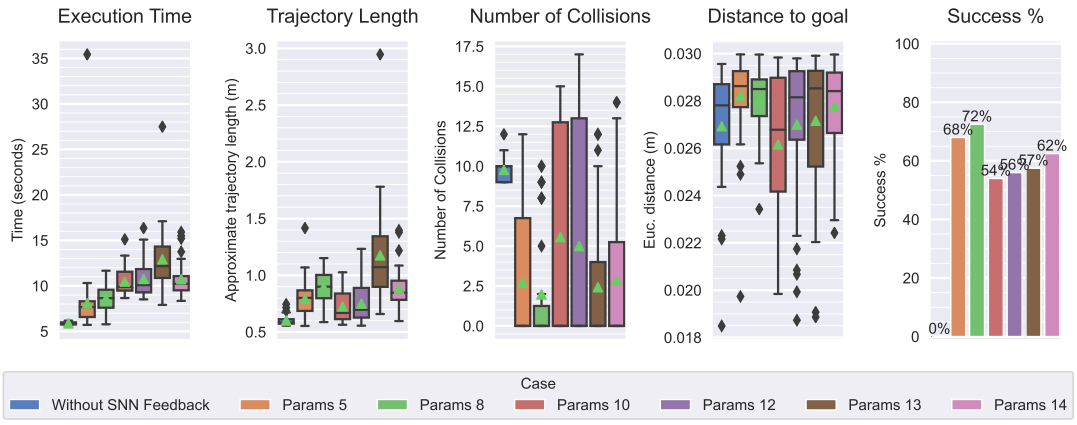


Figure 6.11: Quantitative metric results for validation scenario 4: without SNN feedback and parameter sets 1-12.

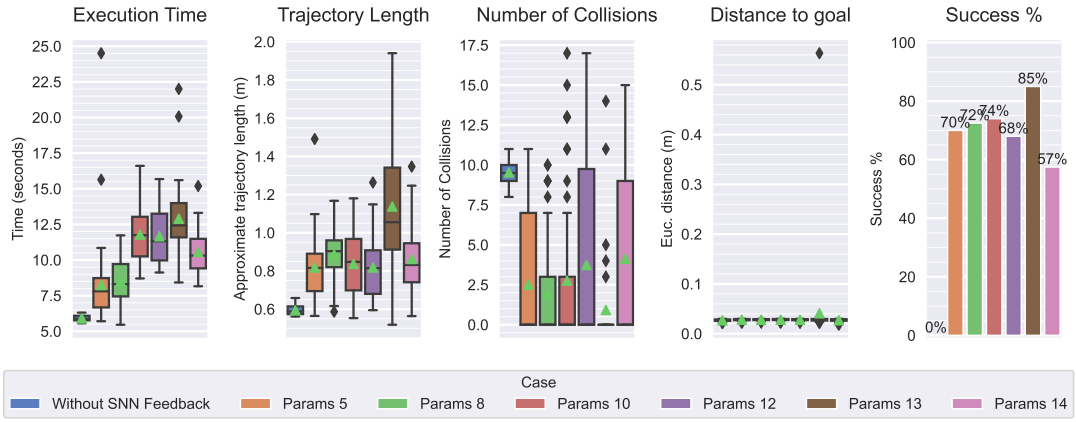


Figure 6.12: Quantitative metric results for validation scenario 5: without SNN feedback and parameter sets 1-12.

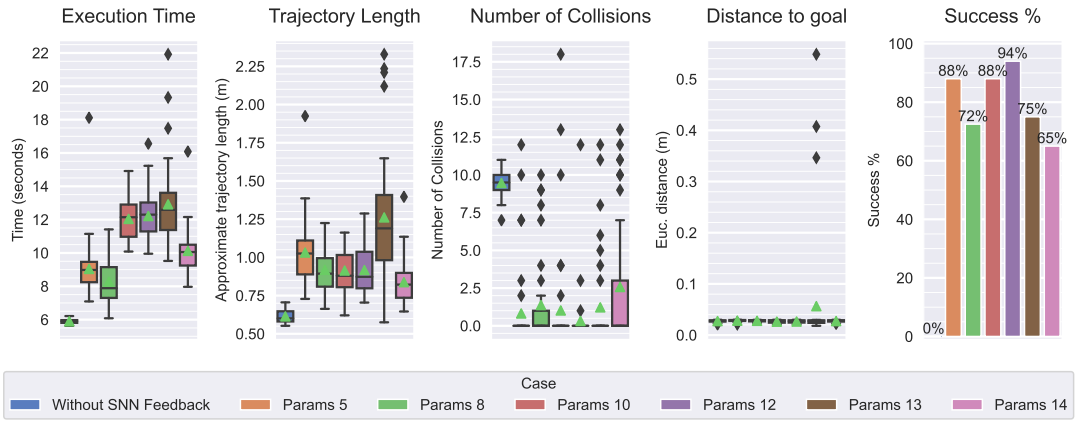


Figure 6.13: Quantitative metric results for validation scenario 6: without SNN feedback and parameter sets 1-12.

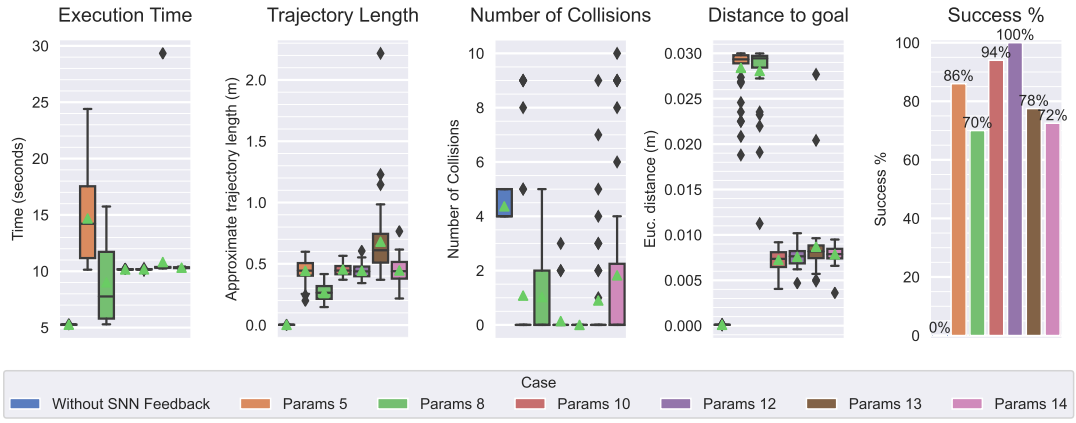


Figure 6.14: Quantitative metric results for validation scenario 7: without SNN feedback and parameter sets 1-12.

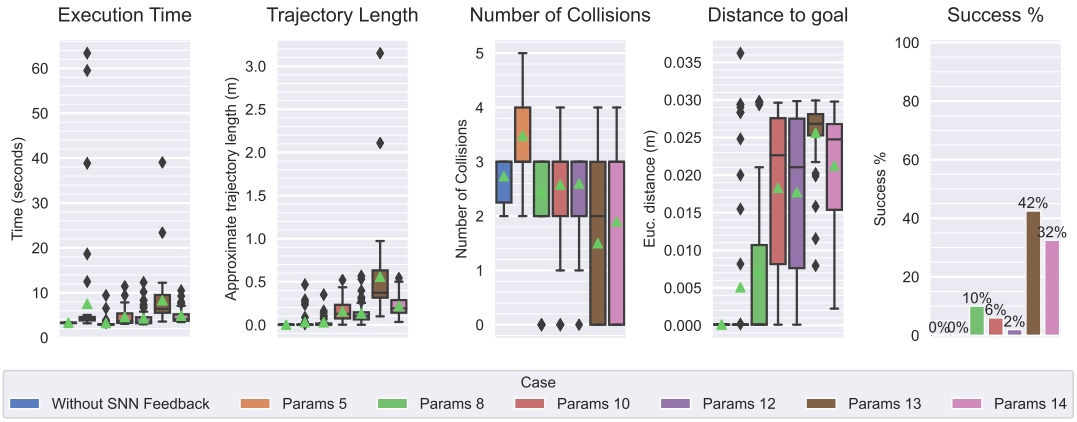


Figure 6.15: Quantitative metric results for validation scenario 8: without SNN feedback and parameter sets 1-12.

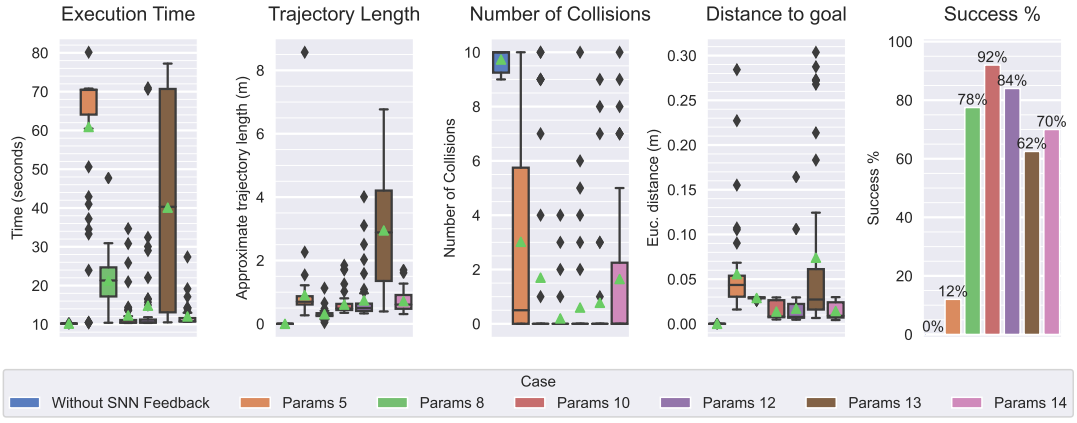


Figure 6.16: Quantitative metric results for validation scenario 9: without SNN feedback and parameter sets 1-12.

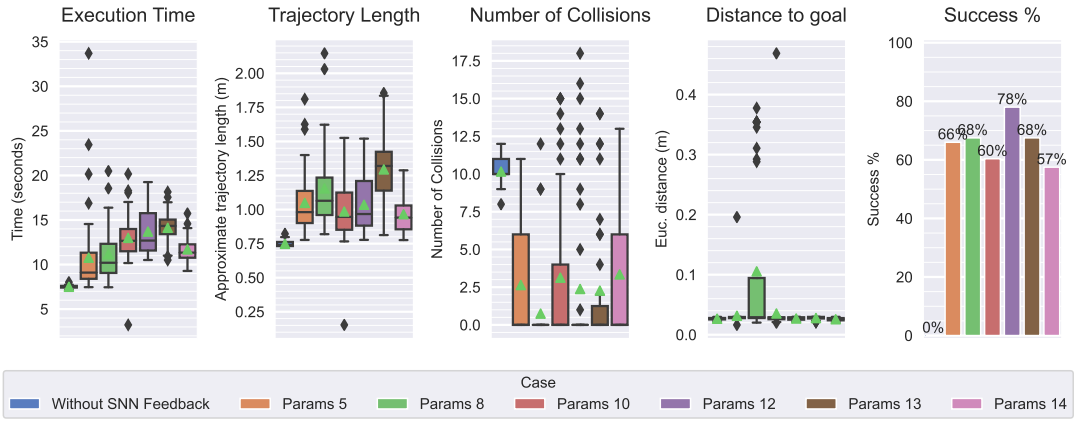


Figure 6.17: Quantitative metric results for validation scenario 10: without SNN feedback and parameter sets 1-12.

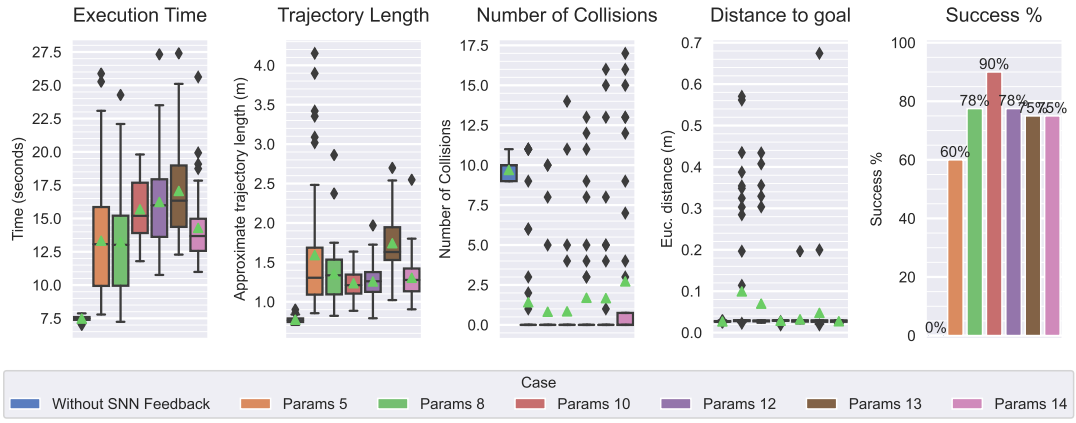


Figure 6.18: Quantitative metric results for validation scenario 11: without SNN feedback and parameter sets 1-12.

6.1.4 Testing Results

From the results of the validation phase we selected parameter set 12 for the ultimate simulation experiments. This set represents the final product of the tuning and validation phases, and is selected based on superior average performance and what we consider the best overall properties from the preceding analysis. As mentioned, the following experiments involve running the best-performing parameter set on all 20 randomly-sampled scenarios of the testing set. Following the same procedure in this phase, we ran batches of $N_{trials} = 30$ trials "without SNN feedback" (control case) and $N_{trials} = 40$ trials "with SNN feedback" (parameter set 12), on each scenario (12-31). Therefore, this phase consisted of 1400 trials. Figure 6.19 shows the quantitative results obtained from these batches of trials in each scenario, comparing the SNN-based obstacle avoidance module's performance against the benchmark control case[9]. Table 6.3 contains the mean values of each metric in each scenario for the "with SNN feedback" case, along with the task mean of each metric: the averages of these values across scenarios of the same task[10].

Quantitative Metric Results

Across all task 1 scenarios (12-16), our SNN-based obstacle avoidance implementation succeeds in 92.5% of trials on average. This performance is fairly consistent over the 5 different task conditions, with the success rate having a low standard deviation (3.5%) and the distance-to-goal, trajectory length, and execution time exhibiting similarly low variability, as is evident from Table 6.3. This includes the novel, low-light setting of scenario 14, which presented no particular difficulties; obstacle avoidance was 90% successful. The executed trajectories also validate these success rates (see Figures D.3a–D.3e in Appendix

[9]Some terms in the scenario specifications provided in the figure captions have been abbreviated, such as Tx for "Task x", Y-B for "Yellow-Black", and Med. for "Medium" (speed).

[10]Note that, in the presented data, a higher average number of collisions does not necessarily translate directly to a lower success rate, when comparing separate batches, if the large number of collisions within a given batch is concentrated in a few trials. This is because these particular trials are considered a failure (success = 0), regardless of how many collisions have occurred. Conversely, a low average number of collisions may cause a very low success rate if the collision instances are spread out between trials. Therefore, the binary metric does not capture the extent of a particular failure, for which the number of collisions is a useful measure when conducting a deeper analysis of performance.

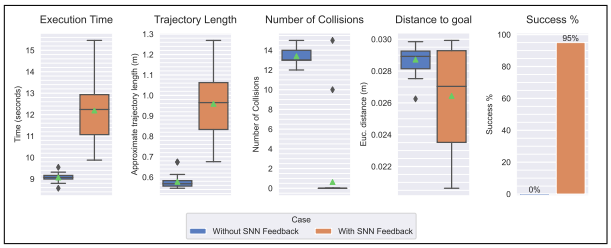
D). Note that the distance to the goal is consistently ∼0.027m since executions are stopped upon reaching the goal-reaching tolerance, $\delta_{\mathbf{g}} = 0.3m$.

For the novel task 2, performance varied significantly more across different task conditions, with a mean and standard deviation of success of 61% and 22.6%, respectively. Most failures were observed in scenarios 17 and 18 (42.5% and 32.5%), which are in the "Office" environment. As expected, we see a similar degradation in obstacle avoidance effectiveness with this background as in the validation trials, for the reasons described in the preceding section. The relatively low mean trajectory length and execution time in these scenarios indicate that the end-effector traveled less, probably not enough to clear the obstacle, due to less effective neural activations. In the remaining scenarios, the arm succeeded more often (77.5%, 85%, and 67.5%). Among these, the lowest success of scenario 21 may be attributed to the obstacle type, "Spiky sphere", which seems to be more challenging than the "Box" obstacle of scenarios 19 and 21. While these results may be acceptable since the task was new, performance was significantly worse than in task 1 and may be improved with feedback from these scenarios for parameter tuning. In general, obstacle speed did not seem to be as significant a factor here as in the task 4 validation scenarios.

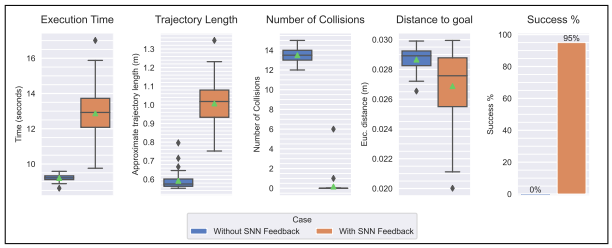
The success rate in task 3 scenarios (22-26) ranged from 57.5% (23) to 85% (25), averaging at 74.5% with a standard deviation of 11%. While the end-effector always reaches the goal, as in task 1, the numbers of collisions seem to vary significantly across trials of a given scenario, and we find insufficient evidence for correlations between the different task variable values, such as the particular background or obstacle color/texture, and the frequency of failures. The difficulty in identifying patterns in the results may be attributed to the following. Occasionally, initial trajectory adaptations would move the end-effector to a region closer to the obstacle from which further adaptations were unlikely to have enough time to steer it away from the obstacle before colliding, even for sufficient perceptual stimuli and neural activation. These occurrences indicate a degree of uncertainty in the resultant trajectory which can be expected from the cascade of non-linear operations performed within the pipeline. Although this is not apparent in other tasks, the emergent avoidance behaviour in these executions appears to be prone to unlucky initial motions, which may explain the inconsistent failures due to collisions. It is also worth mentioning that this task is characterized by a larger obstacle, which is a factor in increasing the likelihood of collisions.

In task 4 experiments, the reported mean success of 66% is significantly skewed due to a complete failure in the high-speed obstacle scenario 28; this is apparent from the median value of 92.5%. This failure is expected from validation phase findings, in which we demonstrated some success in tuning parameters to perform better in a high-speed scenario. However, we concluded that this improvement in performance came at the cost of predictability and safety, and currently consider the inability to cope with very fast obstacles a fundamental limitation in the approach. In other low to medium speed cases, we achieve comparatively high success rates of 92.5-95%, except for scenario 29 (47.5%). Generally, instances of high distance-to-goal values, which were observed in the initial tuning phases, have been greatly diminished through parameter refinement. This is particularly evident from the results of this task.

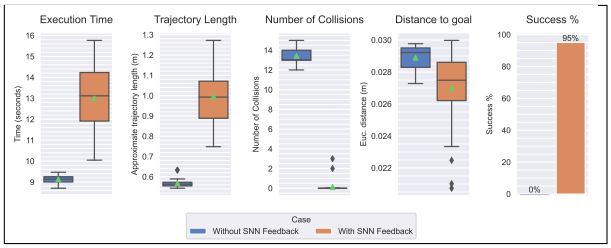
Next, we analyzed the magnitudes of instantaneous accelerations and velocities, measured in executions in all scenarios, particularly comparing the distributions of the measurements between the two cases. Generally, we found that the values were nearly identically distributed between the different scenarios of



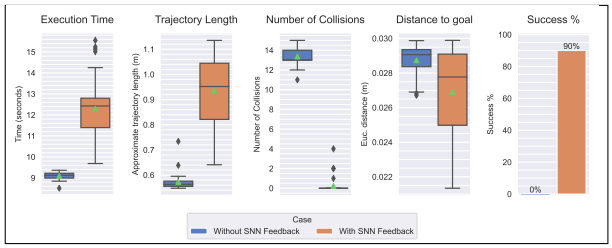
(a) Scenario 12: {T1, Empty, Yellow, Spiky Sphere}



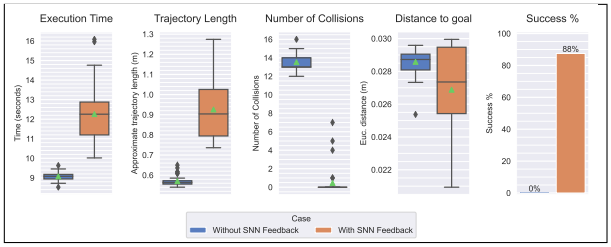
(b) Scenario 13: {T1, Store, Brick, Buckyball}



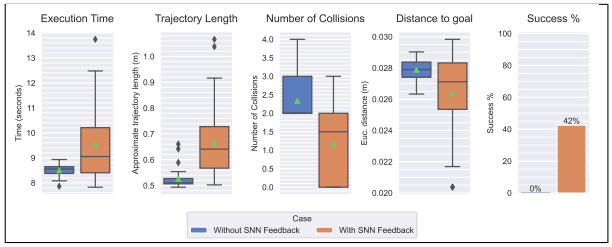
(c) Scenario 14: {T1, Kitchen, White, Buckyball}



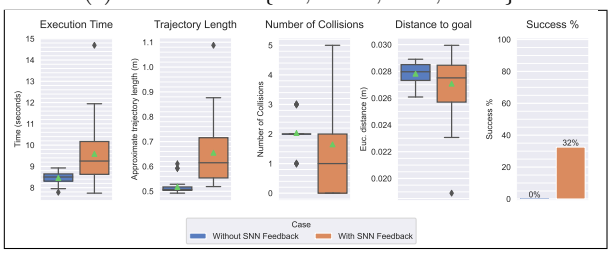
(d) Scenario 15: {T1, Empty, Yellow-Black, Box}



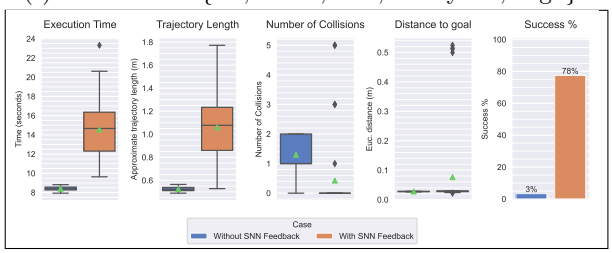
(e) Scenario 16: {T1, Store, Y-B, Rock}



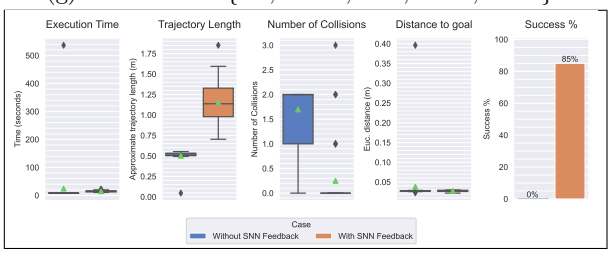
(f) Scenario 17: {T2, Office, Red, Buckyball, High}



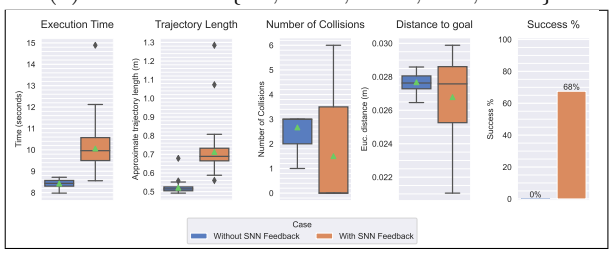
(g) Scenario 18: {T2, Office, Red, Rock, Med.}



(h) Scenario 19: {T2, Store, Brick, Box, Med.}

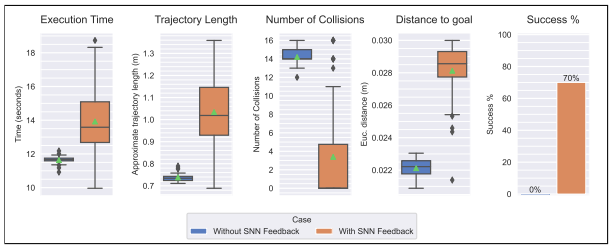


(i) Scenario 20: {T2, Kitchen, Y-B, Box, High}

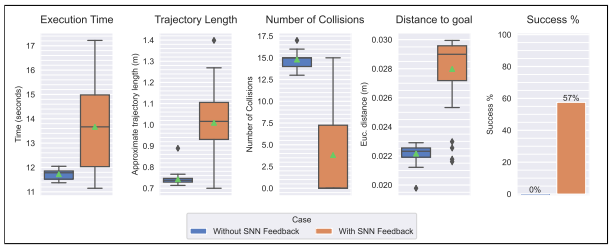


(j) Scenario 21: {T2, Empty, Red, Spiky Sphere, Med.}

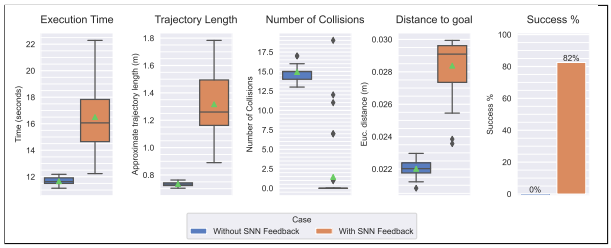
Figure 6.19: Quantitative metric results without vs. with SNN feedback: testing scenarios 12-21.



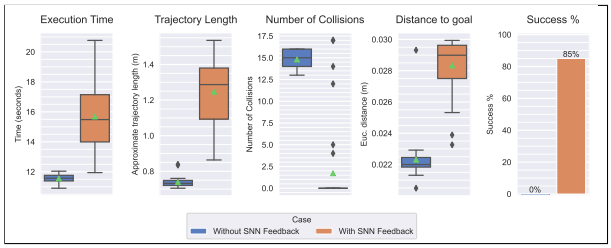
(k) Scenario 22: {T3, Empty, Red}



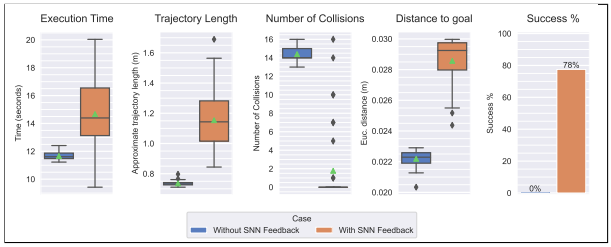
(l) Scenario 23: {T3, Empty, Y-B}



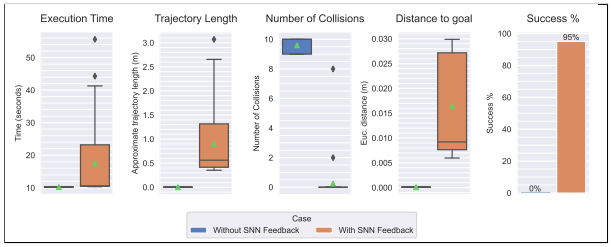
(m) Scenario 24: {T3, Kitchen, White}



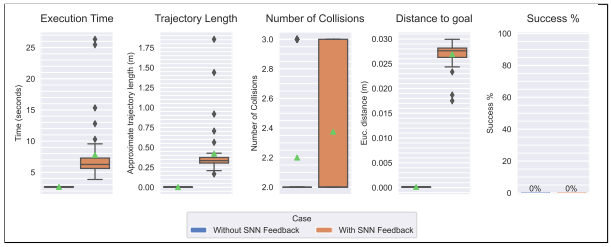
(n) Scenario 25: {T3, Kitchen, Red}



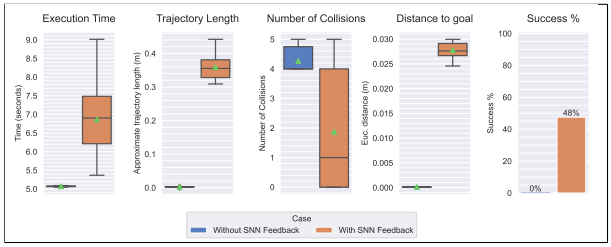
(o) Scenario 26: {T3, Store, Red}



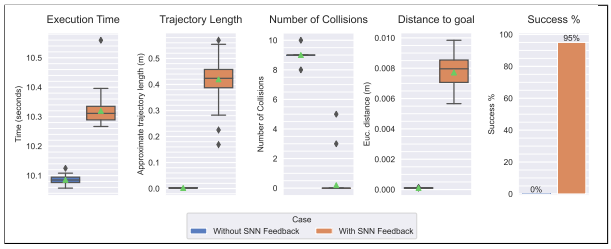
(p) Scenario 27: {T4, Store, Y-B, Buckyball, Low}



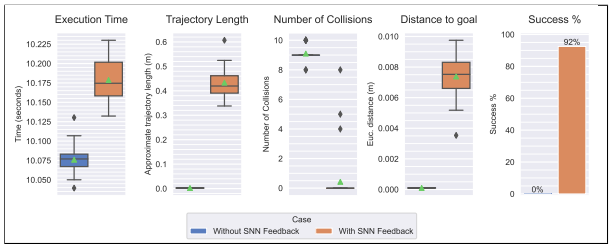
(q) Scenario 28: {T4, Store, Red, Buckyball, High}



(r) Scenario 29: {T4, Empty, Y-B, Spiky Sphere, Med.}



(s) Scenario 30: {T4, Empty, Y-B, Box, Low}



(t) Scenario 31: {T4, Empty, Brick, Buckyball, Low}

Figure 6.19: Quantitative metric results without vs. with SNN feedback (cont.): testing scenarios 22-31.

Table 6.3: Quantitative results of the testing phase runs, averaged over the N_{trials} trials of each scenario. The means across the scenarios of each task are reported at the bottom of each section of the table.

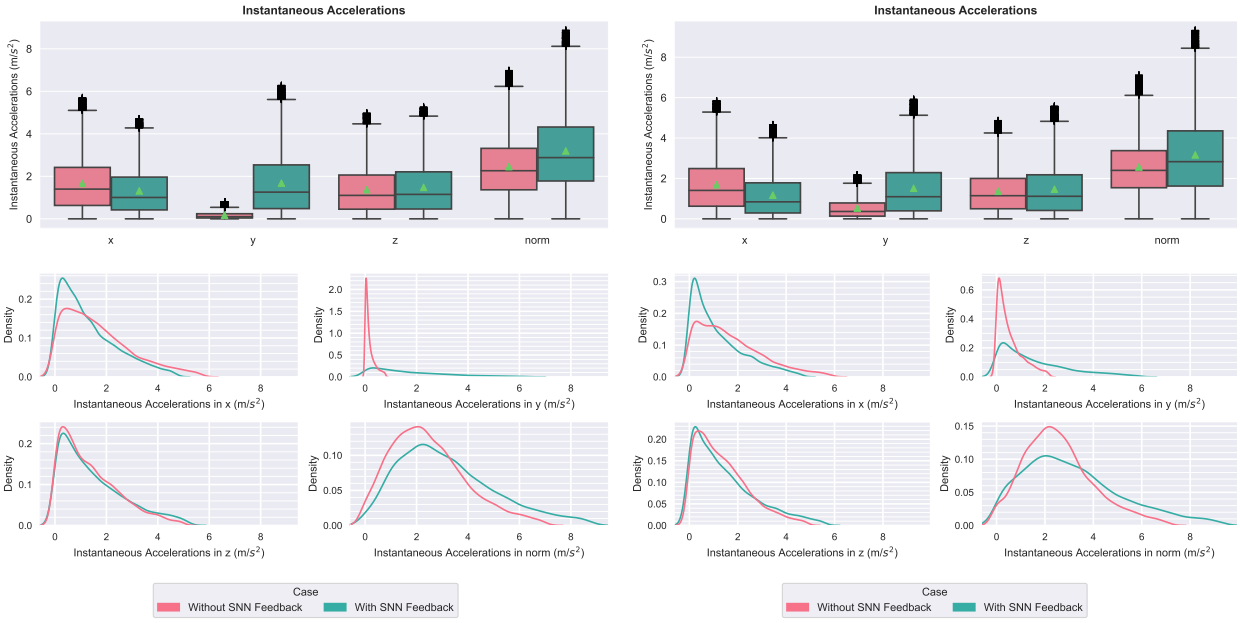
		Results (averages over N_{trials} trials)				
	Scenario No.	Success (%)	Collisions	Dist. to Goal (m)	Traj. Length (m)	Execution Time (s)
Task 1	12	0.950	0.625	0.026	0.965	12.200
	13	0.950	0.175	0.027	1.008	12.875
	14	0.950	0.125	0.027	0.995	12.997
	15	0.900	0.225	0.027	0.936	12.297
	16	0.875	0.450	0.027	0.926	12.260
	Task Mean:	**0.925**	**0.320**	**0.027**	**0.966**	**12.526**
Task 2	17	0.425	1.175	0.027	0.665	9.524
	18	0.325	1.650	0.027	0.655	9.590
	19	0.775	0.425	0.076	1.061	14.546
	20	0.850	0.250	0.028	1.157	14.734
	21	0.675	1.500	0.027	0.713	10.063
	Task Mean:	**0.610**	**1.000**	**0.037**	**0.850**	**11.691**
Task 3	22	0.700	3.400	0.028	1.034	13.924
	23	0.575	3.825	0.028	1.010	13.670
	24	0.825	1.450	0.028	1.317	16.500
	25	0.850	1.725	0.028	1.247	15.690
	26	0.775	1.775	0.029	1.154	14.674
	Task Mean:	**0.745**	**2.435**	**0.028**	**1.152**	**14.892**
Task 4	27	0.950	0.250	0.016	0.901	17.547
	28	0.000	2.375	0.027	0.421	7.780
	29	0.475	1.875	0.028	0.359	6.865
	30	0.950	0.200	0.008	0.420	10.320
	31	0.925	0.425	0.007	0.433	10.179
	Task Mean:	**0.660**	**1.025**	**0.017**	**0.507**	**10.538**

the same task, except for a few notable differences in the dynamic tasks (2 and 4). Since results were fairly consistent within each task type, we present in Figures 6.20 and 6.21 plots for a sample scenario from each task[11]. The observed exceptions to this consistency were as follows:

- Task 2 (scenarios 17-21): accelerations in y were higher in scenarios 19, 20, and (to some extent) 21, compared to 17 and 18. For example, the mean in 17 was $\sim 1 m/s^2$ compared to $\sim 1.6 m/s^2$ in 19 (plotted in Figure 6.20b). Similarly, but to a lesser degree, velocities in 19 and 20 were on average higher than 17 and 18 (but less than a $<0.01 m/s$ increase). This confirms the aforementioned slight inhibition in motion in the "Office" environment (in 17 and 18), which was hypothesized to lead to more instances of not clearing the obstacle in time before a collision.
- Task 4 (scenarios 27-31): accelerations were higher in 28 and 29 compared to the other three scenarios, with the difference mostly contributed by the component in y. For example, the overall acceleration (norm) was $\sim 1.2 m/s^2$ in 27 and $\sim 2 m/s^2$ in 28. A similar trend was observed in velocity values, with the y velocity in 28 being $\sim 0.015 m/s$ higher than in 27. In this case, the two scenarios inducing higher values contain medium to high speed obstacles, as opposed to low-speed obstacles in the others; this shows that more rapid reactionary obstacle avoidance maneuvers are attempted, even though they may not eventually succeed.

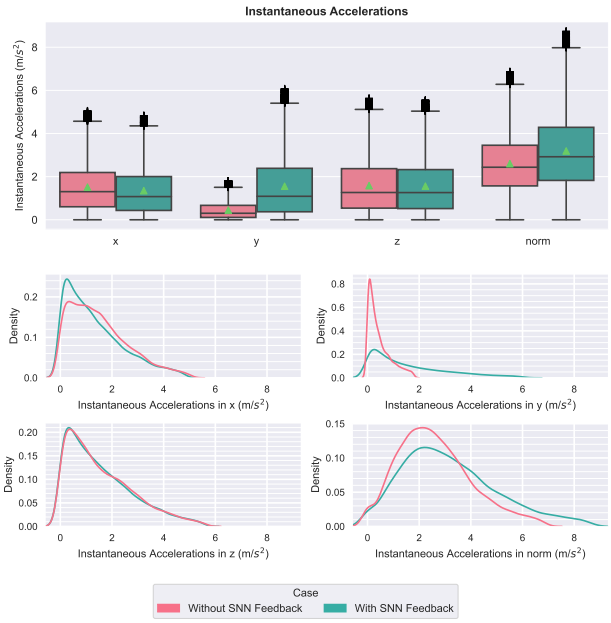
When comparing the "without SNN feedback" to "with SNN feedback" cases in tasks 1-3, we observe generally similar distributions of overall accelerations and velocities. However, magnitudes are often marginally higher when the obstacle avoidance module is used, particularly in the y dimension. This is intuitive, since obstacle avoidance maneuvers would require additional and rapid accelerations to adapt trajectories for successful avoidance. We can infer a preference for side-ways motions in the y direction[12], which is also expected due to the effective avoidance velocity vectors being computed from the camera's image space, ruling out motions in x[13], as expressed in section 4.1.5; even though motions in z[14] are possible, the PF computations appear to produce vectors with smaller z components. The fact that the input image is wider than it is high may have exerted such an influence on the PF gradient computations, but this must be investigated further. Interestingly, accelerations and velocities in x tend to be lower when the obstacle avoidance is utilized. This indicates that the trajectory adaptations naturally slow down forward motion when avoiding obstacles in the tested tasks, which is a desirable effect when aiming to safely clear an obstacle while continuing progress towards a goal. If there were no reductions in forward velocities while executing obstacle avoidance maneuvers, we could expect more collisions simply due to the end-effector continuing to move forward without having sufficiently moved horizontally away from the obstacle. Note that the control case in task 4 exhibits zero magnitudes, since the end-effector simply stays in the set-point position with no obstacle avoidance behaviours. Overall, the similarity of accelerations and velocities to nominal executions while successfully avoiding obstacles and reaching the goal is a favourable property of the SNN-based obstacle avoidance module.

[11] Task 1: Scenario 14, Task 2: Scenario 19, Task 3: Scenario 23, Task 4: Scenario 28
[12] Left-right axis, relative to camera and end-effector
[13] Front-back axis, relative to camera and end-effector
[14] Up-down axis, relative to camera and end-effector

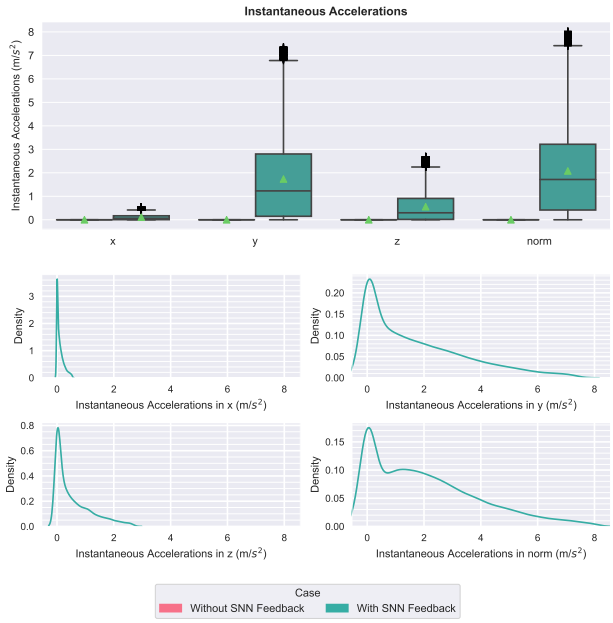


(a) Scenario 14 (Task 1) Accelerations

(b) Scenario 19 (Task 2) Accelerations

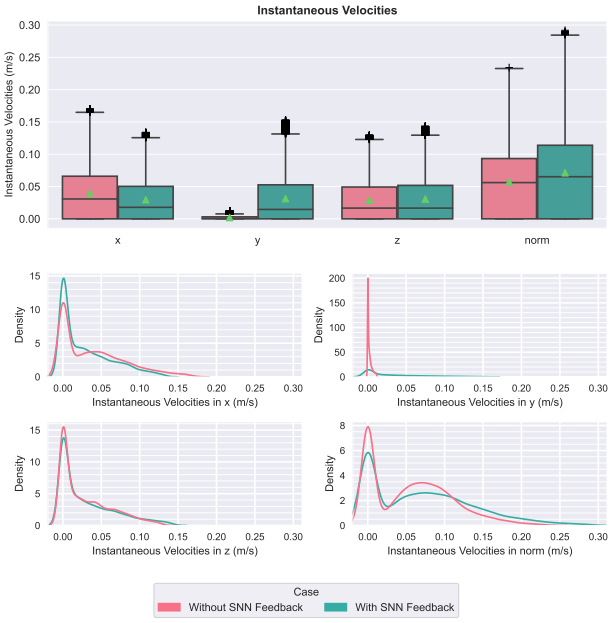


(c) Scenario 23 (Task 3) Accelerations

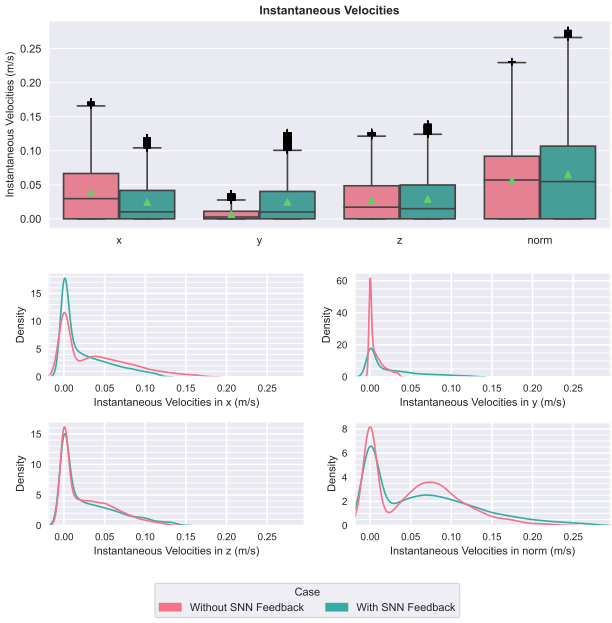


(d) Scenario 28 (Task 4) Accelerations

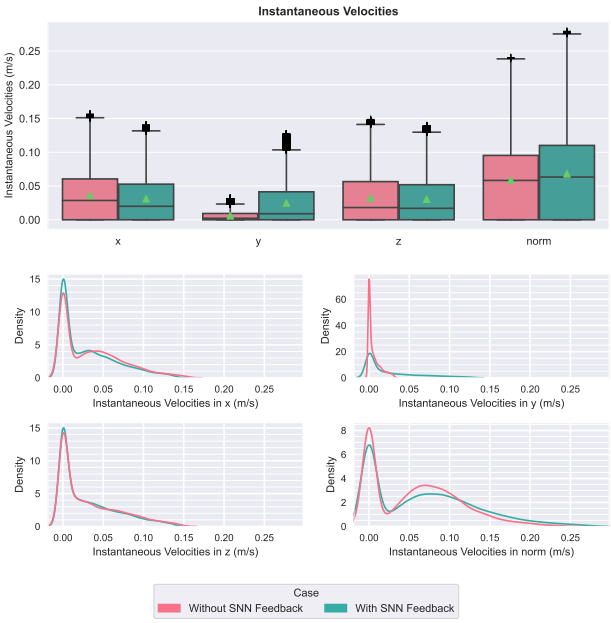
Figure 6.20: Distributions of instantaneous accelerations (top) and their estimated densities (bottom) in each spatial dimension, measured during testing set trials. The figure contains data from one scenario of each task type.



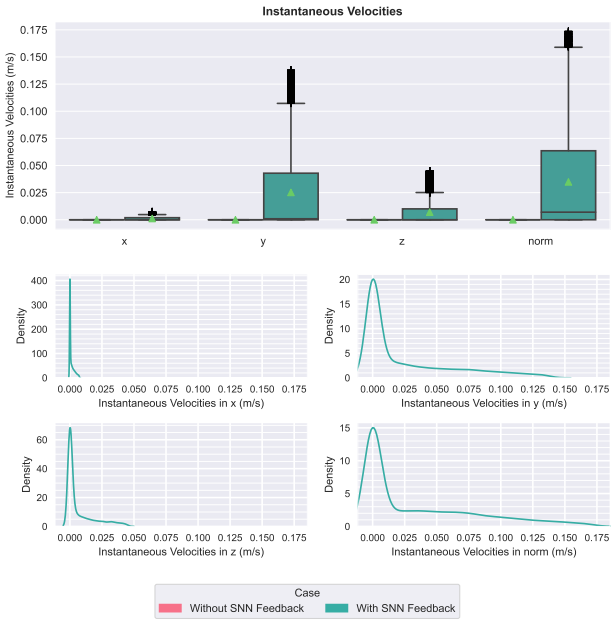
(a) Scenario 14 (Task 1) Velocities



(b) Scenario 19 (Task 2) Velocities



(c) Scenario 23 (Task 3) Velocities



(d) Scenario 28 (Task 4) Velocities

Figure 6.21: Distributions of instantaneous velocities (top) and their estimated densities (bottom) in each spatial dimension, measured during testing set trials. The figure contains data from one scenario of each task type.

Computation Time

The final quantitative performance metric, computation time, is a measure of the algorithmic performance of the pipeline and its primary stages, which is helpful as a benchmark in comparisons and future extensions, in addition to gaining insights about the sources of latencies, if any. For this analysis, we divided the computational stages of the pipeline into:

1. Event emulation
2. SNN simulation
3. Computing obstacle avoidance accelerations (including decoding SNN outputs)

We computed the average time that elapses in a single iteration of each stage, by recording single iteration durations in four trial executions (each execution yielded ∼50 measurements, i.e. approximately $50 \times 4 = 200$ measurements per stage) and calculating the mean for each stage. The sum of these computation times provides an estimate of the total computation time required to produce an obstacle avoidance velocity command (output) from the source images (input). In addition, we also measured the computation times of the first two stages when no stimulus is presented, i.e. the arm is stationary and no motion is induced in the image, to study the hypothesized dependence of SNN computations on the magnitude of the input. These values are presented in Table 6.4.

It is important to note that the latencies indicated by these values are not actually observed in executions as would be in a fully sequential pipeline. Instead, the ROS implementation is designed to run these stages in parallel, where the message handling protocol ensures that each component processes the most recent input, which is often available before the component finishes its preceding computation. Nevertheless, the presented computation times provided an understanding of the time required to perform computations.

The values in Table 6.4 indicate that most time is spent in simulating the SNN, which is primarily handled by the external BindsNET package, while the least is expended in the decoding of SNN output and computation of avoidance accelerations. In total, the pipeline computations require ∼0.15s in a single pass[15]. While this performance was sufficient in producing successful avoidance behaviours, the implementation has not been optimized for computation time and could be improved in future extensions. It is also worth mentioning that this is currently implemented in Python and relies on external Python packages (such as BindsNET and pydmps); optimized implementations of the required functionalities may provide speed improvements. In addition, note that the event emulation stage becomes unnecessary when switching to a real EC.

A particularly interesting observation from Table 6.4 is the reduction in SNN computation time when no motion stimulus is present, i.e. no input spikes are induced. This indicates that the amount of computations is positively correlated with the number of events in the image, which is effectively a measure of new information in the scene[16]. If we consider the changes in the perceptual input and their locations to be the relevant information for the intended behaviour, which holds for our obstacle avoidance

[15] The time it would take to react, if the implementation required sequential processing

[16] In a separate test, the computation time was measured during varying degrees of motion in the image. We found that the computation time seemed to indeed be proportional to the density of events, which is in turn proportional to the amount of motion.

Table 6.4: Mean computation times (in seconds) of the three main stages of the SNN-based obstacle avoidance pipeline and their total. Measurements of the first two stages when no stimuli are presented are also included.

Stage	Computation Time (s)	
	During Executions	No Stimuli
Event Emulation	0.025 ± 0.001	0.025 ± 0.001
SNN Simulation	0.123 ± 0.014	0.086 ± 0.007
Avoidance Acceleration Computation	0.002 ± 0.001	-
Total	0.150 ± 0.016	0.111 ± 0.009

problem, then we can say that the SNN exhibits a property of expending computations and time only to process relevant and salient information (in the context of the task). This is enabled by utilizing event data as the input, since events efficiently capture this information, while the SNN for its part does not perform unnecessary computations whenever there are no spikes (derived from events), for example by computing or propagating spikes through the network. This provides evidence for the input-dependent computational property of the SNNs, which is different from conventional DNNs, where every pixel intensity value in an input image is processed and every neuron necessarily activated at every timestep, even for a completely static input signal. In some applications, such as obstacle avoidance, this property has the potential to provide comparatively better power and time efficiency.

Qualitative Results

As part of the final analysis in simulation experiments, (the testing phase), we evaluate executions based on our qualitative criteria. (Refer to section 5.1.3 for detailed interpretations of each.)

We consider the extent to which our approach is reliable to be correlated with how consistently it produces positive results in the same task conditions, which we equate to its rate of success in each scenario (i.e., the fraction of successful trials in imminent collision scenarios). In particular, a high value indicates that obstacle avoidance was consistently successful within N_{trials} executions, while a lower value indicates an inconsistency that shows unreliability, even though some success was achieved. From Table 6.3, we observe that success varied significantly *across* different task conditions. However, the overall rate of success averaged over each of the consistent task conditions[17] was about 74%, even when including the scenario 28 outlier. This was determined to be the only outlier among the scenario success rates using the same inter-quartile range-based outlier detection method described by equation 5.5. The median success rate, which is not as influenced by the outlier, was 84%, and the obstacle avoidance module had higher success rates in half of the test set scenarios. Broadly speaking, we can therefore say that the implementation and chosen parameter set score **moderately** on reliability. Looking at individual scenarios, we can more specifically conclude that the implementation is:

[17]The mean over all success values in the first column, i.e. 20 data points.

- not reliable in high-speed scenarios (at least for the objects we consider, traveling at approximately $0.36m/s$),
- very reliable in task 1 scenarios
- more reliable in static than dynamic obstacle scenarios
- seemingly less reliable in the "Office" environment (possible due to less contrasts and more clutter)[18]
- as reliable in high ("Store") and low ("Kitchen") illumination conditions

We compute and plot the magnitudes of the derivatives of angles that describe directional changes of the end-effector, $\dot{\zeta}$, in order to evaluate predictability. The estimated densities of these values are displayed in Figure 6.22. Here, we randomly sampled a representative scenario from each task and computed the results from its data. Note that, similar to the case of the velocity and acceleration plots, we used the inter-quartile range method of Equation 5.5 to remove outliers from the recorded measurements which may disproportionately skew the distributions[19]. The plots indicate that the inlying values of $\dot{\zeta}$ are similar between the cases and are generally fairly low, being mostly concentrated in the [-2,2] deg/s range in the first three tasks. This suggests that the evolution of the end-effector's trajectory, i.e. the next positions it would travel towards, would be generally easy to predict when observing its motion; a conclusion that has been verified from visual observations. In task 4[20], the values indicate very low magnitudes of $\dot{\zeta}$; therefore, motions would theoretically be even easier to predict in these scenarios. Overall, these results indicate a high level of predictability in the motions produced by the SNN-based obstacle avoidance implementation. It bears mentioning that some instances of high $\dot{\zeta}$ values were found in measurements, i.e. the filtered outliers; however, these were infrequent to the degree that they represented significant outliers.

In order to assess how safe the executed obstacle-avoiding trajectories are, we examine the distributions of end-effector velocities, presented in Figure 6.21. For a reference point, we refer to the ISO standard for industrial robot safety requirements (ISO 10218-1:2011 (2011))[21], which suggests a threshold for tool speed under which a robot can be considered to operate in a safety-rated, "reduced speed control" mode: $0.25m/s$ (Beckert et al. (2017)). Such a mode is intended to allow humans enough time to react to a robot's motion in shared workspaces, in case it is deemed potentially dangerous. This speed limit can therefore represent a logical benchmark for evaluating the degree to which executed trajectories are safe. The overall end-effector speeds recorded during simulation experiments had an average value of $\sim 0.07m/s$. From the box plots in the figure, it is evident that the end-effector occasionally approached speeds of approximately $0.29m/s$, but these instances were at the maximum, far beyond the inter-quartile range, and thus were relatively rare. This is also visible in the density plots (bottom right in each figure), which additionally confirm a mode of the distribution at the $0.07m/s$ value[22]. As far as the evidence provided

[18] However, the sampled test scenarios included only a couple with the "Office" background, both of which were of the same task type. Therefore, more tests may need to be conducted to conclusively isolate this background as a cause for less reliability.

[19] In addition, we ensure that the number of points from each case with which the plotted densities are computed is equal, by randomly sampling N_{min} points from each, where N_{min} is the minimum of i) the number of measurements in the "Without SNN Feedback" case and ii) the number of measurements in the "With SNN Feedback" case.

[20] Note that the "Without SNN Feedback" case is not plotted, since all measurements would be 0 in this task.

[21] This standard is to be updated in the ISO/FDIS 10218-1 standard, which was under development at the time of writing.

[22] Note that the dominant mode appears to be at 0.0m/s. This is likely due to the fact that many measurements come from

by this comparison of instantaneous velocities to the ISO standard safety rating suggests, the trajectories produced by the obstacle avoidance module could be considered largely safe. Although an analogous threshold for safe accelerations is not available, the distributions of values in Figure 6.20 indicate that accelerations are not noticeably excessive, particularly when compared to nominal executions.

We focus on the smoothness property of executed trajectories in evaluating how natural they appear. Upon visual inspection of the trajectories, such as from the plots in Appendix D (Figure D.3; particularly visible in the first 5 scenarios) or the sample trajectory plotted in Figure 6.23, it can be determined that the obstacle avoidance trajectories seem fairly smooth, especially given that the algorithm actively optimizes only for the fulfillment of the obstacle avoidance criterion of the task. However, we also conducted an analysis of the jerk profiles of executed trajectories to quantify smoothness (as mentioned in section 5.1.3). While jerk is commonly used as a trajectory optimization criterion (Gasparetto et al. (2015)), we utilize it here as an evaluation criterion by comparing to theoretical minimum-jerk profiles that maximize smoothness. We compare against a theoretical optimum since, to the best of the author's knowledge, no standardized benchmarks of acceptable jerk values or profiles have been presented in the literature. From Fligge et al. (2012) and (originally) Flash & Hogan (1985), a uni-dimensional minimum-jerk trajectory between positions x_i and x_f lasting for a duration of T_d can be generated by the formula:

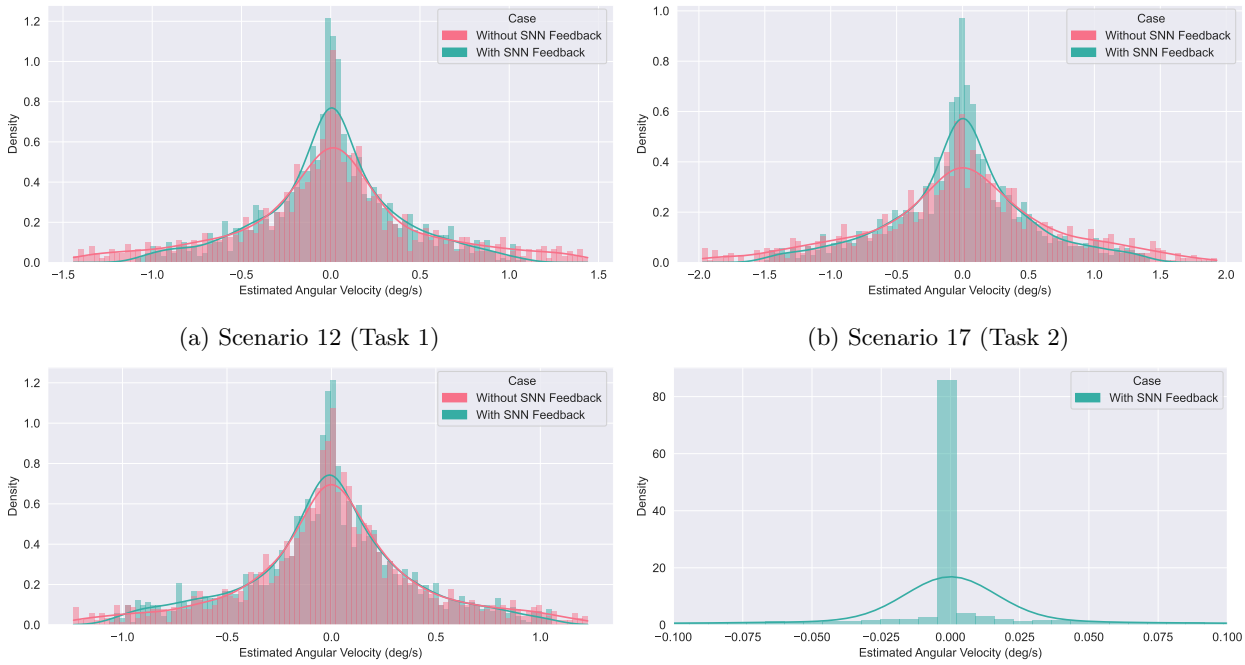
$$x(t) = x_i + (x_f - x_i)(10\left(\frac{t}{T_d}\right)^3 - 15\left(\frac{t}{T_d}\right)^4 + 6\left(\frac{t}{T_d}\right)^5) \tag{6.11}$$

In our analysis, we randomly sampled a trajectory from the scenario 12 batch, shown in Figure 6.23, and extracted a segment of the trajectory in each spatial dimension. For each segment, we generated a theoretical minimum-jerk trajectory spanning the same position value range and time duration, using equation 6.11. Then, we estimated the jerk in each segment by computing the third-order derivative of the positions, and compared the jerk values obtained from the executed trajectory and the theoretical minimum-jerk trajectory in each dimension.

In Figures 6.24–6.26, we plotted the positions and estimated jerk values of the trajectory segment in each case for x, y, and z[23]. In addition, the sub-plots on the bottom right depict estimates of the densities of jerk values in each case, along with the mean values (dotted lines) within that segment. From the absolute jerk values, we find that the executed trajectories generally exhibit higher jerks than the theoretical trajectories, though the difference is not extraordinary and the sample execution occasionally had slightly lower values (particularly in y and z). The density plots also indicate that executed jerks are on average higher and more spread out (i.e. have a larger variance). These results prove that trajectories executed in our tests applied jerks that generally exceed a theoretical minimum, but not to the extent that sudden accelerations occur noticeably frequently and thus produce unnaturally uneven transitions along a trajectory. While significant jerks can be expected when reacting to unforeseen circumstances such as the presence of obstacles, this analysis and a subjective assessment of observed executions suggests that the current approach does not significantly compromise trajectory smoothness. A more exhaustive

portions in the trajectory where the end-effector is slowly approaching the next position (according to the PID controller), and its speed is near zero for a relatively significant amount of time.

[23]The time duration between successive points in these plots is approximately 0.06 seconds.



(a) Scenario 12 (Task 1)

(b) Scenario 17 (Task 2)

(c) Scenario 24 (Task 3)

(d) Scenario 27 (Task 4)

Figure 6.22: Histograms and kernel density estimates depicting the distributions of estimated angular velocities of the end-effector measured during testing set trials. The figure contains data from one scenario of each task type.

evaluation of naturalness could be achieved by conducting a user study to aggregate subjective evaluations of executed trajectories to better approximate perceived naturalness, and by extending the jerk analysis to include more trajectory samples and segments in an aggregate measure of applied jerks.

6.2 Real Robot Experiments

In the real robot experiments, we ran the implementation we developed in the simulation on the real Kinova Gen3 arm in the task scenarios defined in Table 5.7. We particularly examined how well the tuned parameter set (12) transferred to the real platform, and any necessary adaptations. We ran N_{trials} trials in each scenario with the obstacle avoidance module, then assessed similarly assessed performance compared to executions without the module. In these experiments, $N_{trials} = 30$ were executed per scenario.

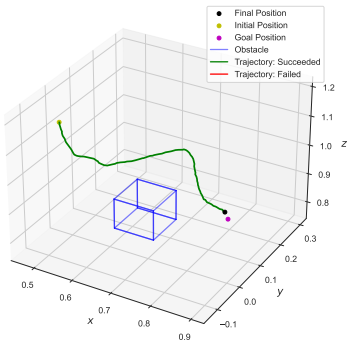


Figure 6.23: Sample trajectory (scenario 12)

From preliminary testing, we found that the parameter set we tuned in simulation performed acceptably, but a slight degradation in motion quality necessitated minor adaptations. In particular, the motions executed by the arm were occasionally somewhat oscillatory and not as smooth as in the simulation. In

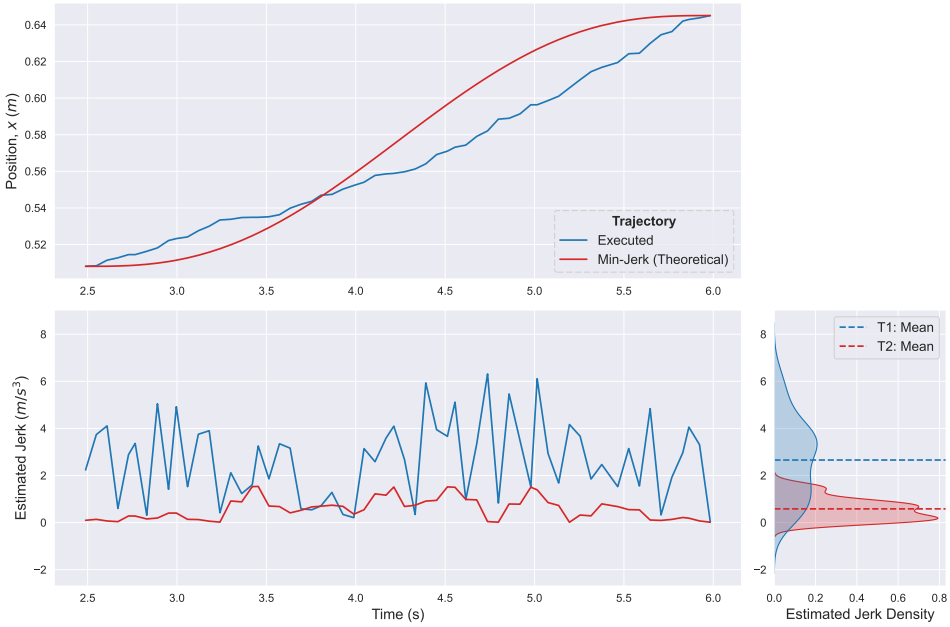


Figure 6.24: A comparison of estimated jerk between a section of the sample trajectory (Figure 6.23) in x (between 2.5 and 6.5 seconds in the execution) and a minimum-jerk trajectory. The plot contains the positions (top), jerks (bottom left), and estimated densities of jerk values (bottom right).

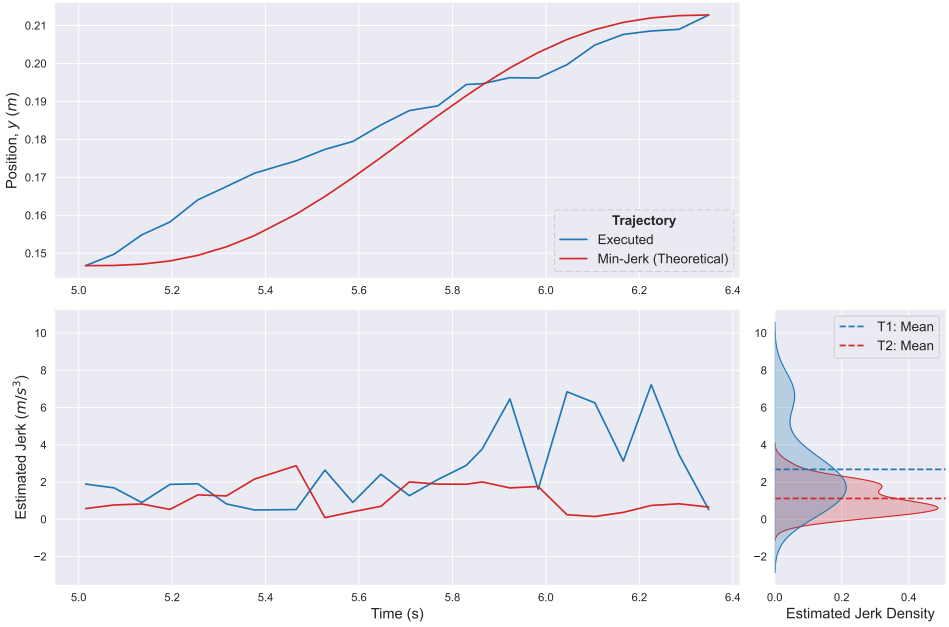


Figure 6.25: A comparison of estimated jerk between a section of the sample trajectory (Figure 6.23) in y (between 5.0 and 6.4 seconds in the execution) and a minimum-jerk trajectory. The plot contains the positions (top), jerks (bottom left), and estimated densities of jerk values (bottom right).

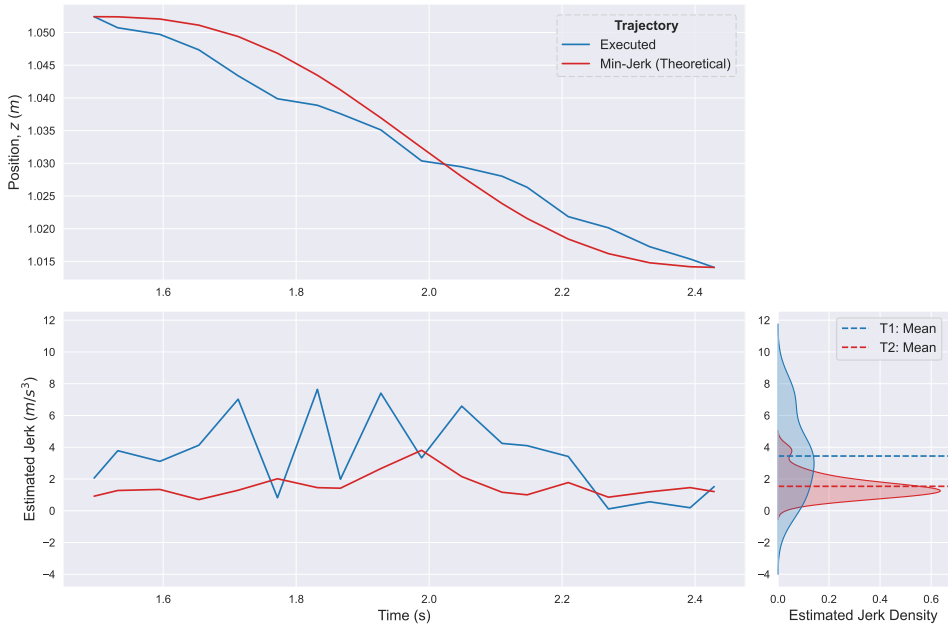


Figure 6.26: A comparison of estimated jerk between a section of the sample trajectory (Figure 6.23) in z (between 1.5 and 2.5 seconds in the execution) and a minimum-jerk trajectory. The plot contains the positions (top), jerks (bottom left), and estimated densities of jerk values (bottom right).

addition, while the obstacle avoidance module was active, the arm would react somewhat drastically to minor motions in the image or to background events caused by ego-motion (even with active filtering). We addressed these issues by adapting the following parameters (out of an effective total of 36 which excludes a few irrelevant parameters, like *source_type*):

- $K_p : 5.0 \rightarrow 2.0$, $K_d : 10.0 \rightarrow 5.0$, $K_i : 0.0 \rightarrow 5.0$
- $\delta_{\mathbf{y}} : 0.01 \rightarrow 0.02$
- $\theta : 28 \rightarrow 45$

The controller gain values were a primary cause for non-smooth motions, which more closely resembled the ones executed in the simulation following re-tuning. This was expected, since the values were tuned for the physical dynamics of the simulation, and these observations confirmed the discrepancies compared to real-world dynamics (mainly in the actuators) that necessitate adapting the motion controller. Secondarily, slightly increasing the position reaching tolerance contributed to smoother motions by easing the criterion that determines when a position along the trajectory is reached; as a result, transitions between positions were less abrupt. Similarly, this highlighted a discrepancy in the actuator dynamics, such as friction effects, between the real and simulated arms, such that the original value was satisfactory in the simulation. The stronger reactions to motion stimuli were likely a result of significant differences in the densities of events generated from simulation as opposed to real-world images. Even in fairly plain backgrounds and different lighting conditions, the event camera emulator generated more events for similar motions and distances to

other objects (compared to the simulation environments), indicating that real camera images contained more colour variations and contrasts (i.e. richer color images) and possibly more noise. By increasing the event emission threshold, θ, we effectively offset this naturally larger variation in RGB data and thus produce more similar event densities and, by extension, SNN activations and resultant avoidance motions to those observed in the simulation.

A notable constraint we had to place during the real robot experiments was to deactivate an auto-focus feature of the RGB camera mounted on the arm. Intuitively, a variable level of focus in consecutive RGB images presents a source of intensity variation that is external to the scene dynamics. Since we derive events from changes in consecutive intensity values and regard this information as representative of the motion in the image, this feature can potentially pollute the data and is thus necessary to eliminate.

Quantitative Metric Results

Figure 6.27 contains the familiar metrics plot, here depicting the quantitative results of the real experiment scenarios. In this case, we plot together the results from each scenario (R1-R4) in addition to a single batch of "Without SNN Feedback" trials[24]. Note the removal of the $N_{collisions}$ metric from consideration. Since it depends on the possibility of counting instances of the end-effector intersecting with the obstacle, it is only possible in simulation and loses its meaning in the real experiments. Instead, a collision simply influences the success metric, such that any contact leads to a failure. Note that, as in the simulation experiments, a trial is considered a failure if the end-effector touches the obstacle at all. Table 6.5 summarizes the results in terms of the mean values of each metric.

The arm was 90% successful in avoiding the wooden block in task 1, with three recorded failures occurring when the end-effector moved too closely to the obstacle after executing an avoidance maneuver

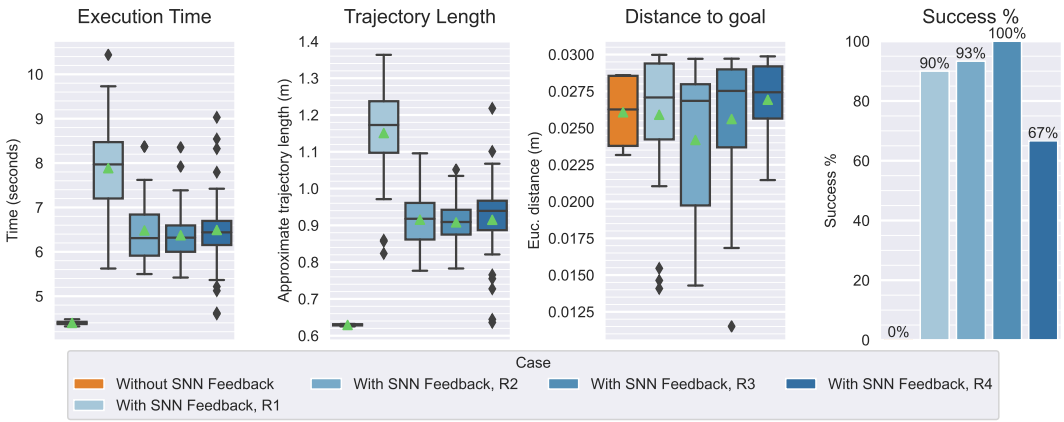


Figure 6.27: Quantitative metric results without vs. with SNN feedback (cont.): real robot experiment scenarios R1-R4.

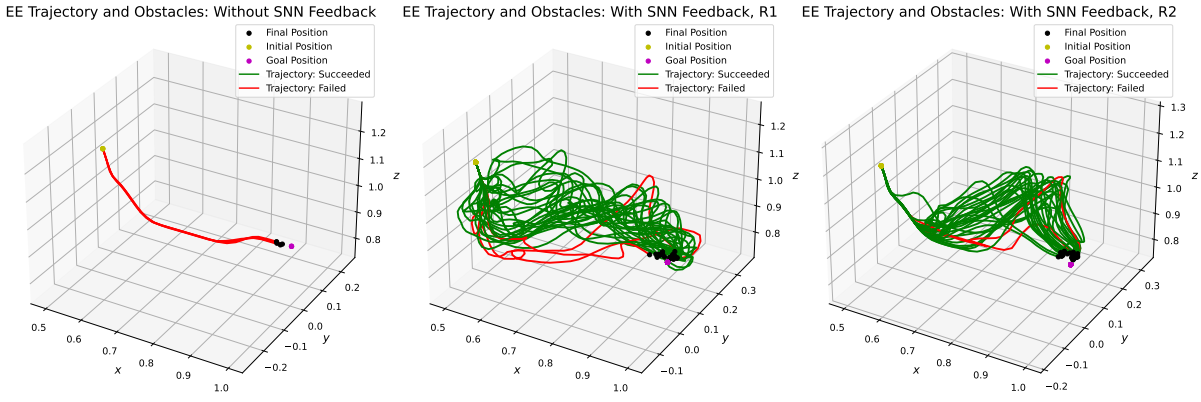
[24]Different from previous results, we aggregate "Without SNN Feedback" executions from each of the scenarios and present the aggregate as a single case, instead of a separate case for every scenario. This is because the statistics are not expected to differ between the scenarios when no obstacle avoidance is used, since we use the same parameter set and the arm always executes the pre-planned motion trajectory.

Table 6.5: Quantitative results of the real robot experiment runs, averaged over the N_{trials} trials of each scenario.

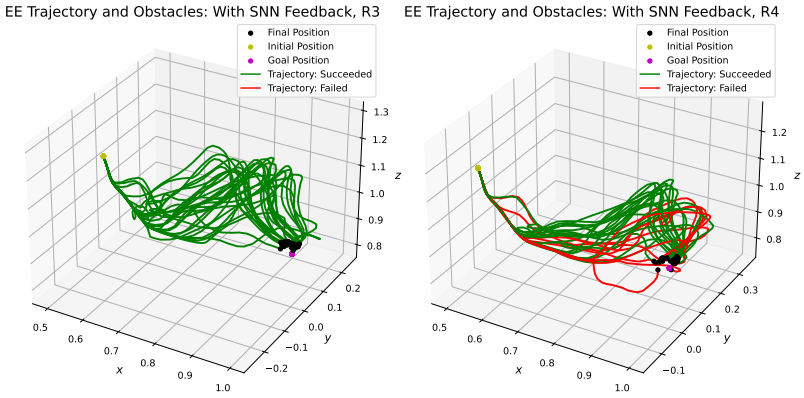
		Results (averages over N_{trials} trials)			
	Scenario ID	Success (%)	Dist. to Goal (m)	Traj. Length (m)	Execution Time (s)
Task 1	R1	0.900	0.026	1.151	7.885
Task 2	R2	0.933	0.024	0.915	6.482
	R3	1.000	0.026	0.908	6.374
	R4	0.667	0.027	0.915	6.492
	Mean:	**0.875**	**-**	**-**	**-**

and just touching it. The results from the three task 2 scenarios showed varying levels of success. The arm was most successful with the metal bar obstacle (R3), as it never failed the task, followed by the hand obstacle with a success rate of 93% and finally the wooden block with 67%. As evident from the distance-to-goal values being very close on average to tolerance $\delta_{\mathbf{g}}$ and having a small variance (see box plots), as well as the trajectory plots in Figure 6.28, the end-effector always reached the goal during these experiments. It follows that all instances of failures were due to collisions. The execution times and trajectory lengths had an expected level of variance that was generally similar to simulation executions, and which is due to some variation in trajectory shapes across executions. Note that, even though the initial and final end-effector poses were the same in both tasks, the task 1 scenario (R1) had consistently higher values for both metrics. This is because, unlike in task 2 (R2-R4), the obstacle is visible from the beginning and so avoidance maneuvers start early in the trajectory; naturally, these maneuvers tend to lengthen the trajectory and extend the time required to reach the goal. Overall, the obstacle avoidance module performed satisfactorily with the minimally-adapted parameter set, succeeding in 88% of all 120 trials.

Interestingly, the higher failure rate of scenario R4 was due to the visual properties of the obstacle in relation to the background. The wooden block obstacle blended well with the table which dominated the background of the image during executions of task 2. This can be seen in Figure 6.30a, which shows two images captured by the onboard camera during an execution of scenario R4. As the obstacle comes into view from the right, it generates less events and therefore less neural activation than the more contrasting hand and metal bar obstacles. This causes the eventual trajectory adaptations to occasionally be too weak or too late to move the end-effector sufficiently away from the obstacle before a collision. Contrarily, the perfect success in R3 could be explained by the significantly higher contrast of the obstacle color to that of the background (see Figure 6.30b). Intuitively, the amount of events generated in the R4 case is less than in R3, since similar colors induce lower intensity differences (and vice-versa), therefore leading to a less effective response from the obstacle avoidance module for less visible objects. It can be argued that this limitation similarly applies in the biological domain, since our ability to see an object seems to

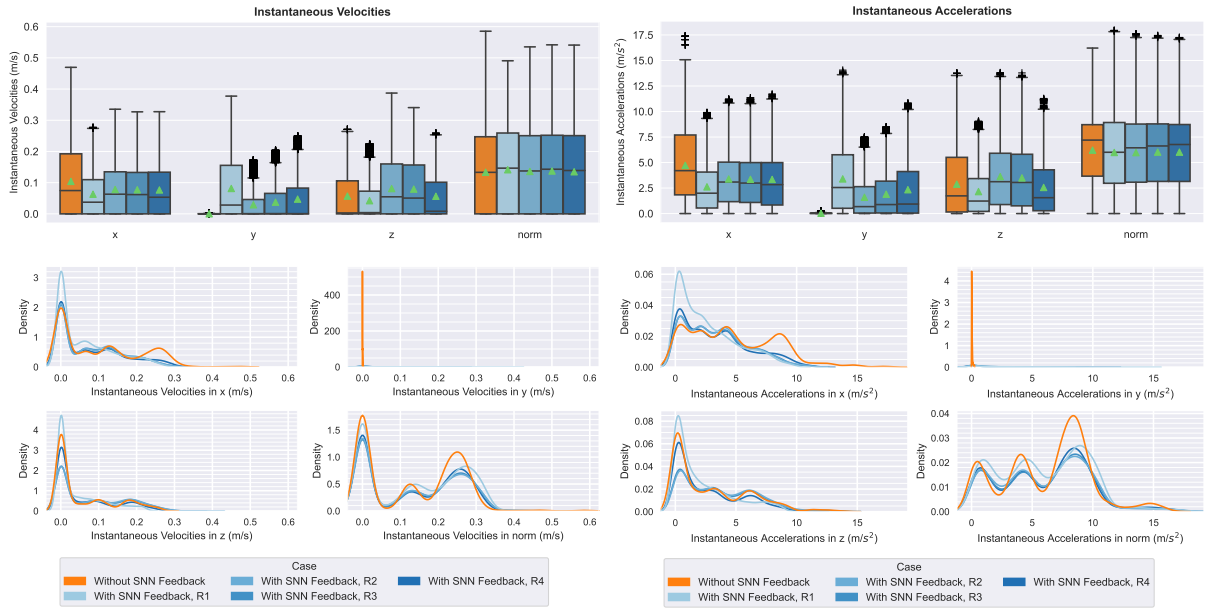


(a) Scenario 12 (b) Scenario 13 (c) Scenario 14



(d) Scenario 15 (e) Scenario 16

Figure 6.28: Trajectories executed in real robot experiments without SNN Feedback and with SNN feedback in scenarios R1-R4.



(a) Velocities

(b) Accelerations

Figure 6.29: Distributions of instantaneous velocities and accelerations in each spatial dimension, measured during real robot experiments. The figure contains data from nominal executions (without SNN Feedback) and scenarios R1-R4.

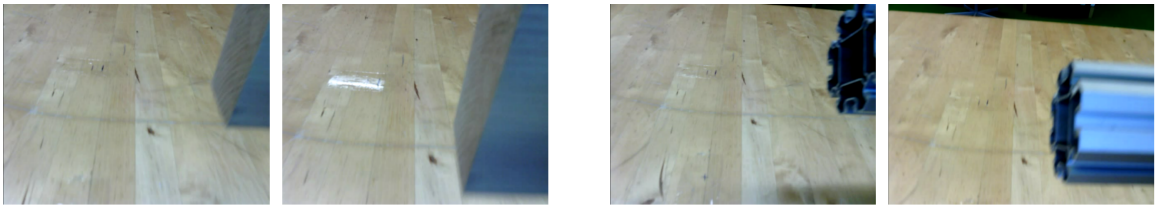
diminish the better it blends with the background[25].

Figure 6.29 shows the distributions of velocities and accelerations recorded in each scenario. On average, these were higher than in the simulation experiments. This is likely due to the values of the motion controller parameters (particularly, the gains) that were selected when adapting to the real robot; while the executed trajectories look qualitatively similar, the instantaneous speeds measured at a low resolution appear to have increased. Moreover, it is possible that inaccuracies in position updates within the simulation could have contributed to the discrepancy in measurements. A more relevant observation is the similarity of velocity and acceleration magnitudes with or without the obstacle avoidance module, and the generally higher values in y when executing avoidance maneuvers; both findings are in agreement with simulation results. In addition, the aforementioned similarity, despite larger overall magnitudes in comparison to simulation experiment data, further suggests that the higher velocities and accelerations are due to the motion controller and its parameterization, and not the obstacle avoidance behaviour.

Qualitative Results

We qualitatively evaluated the trajectories executed during real robot experiments with the same criteria and (interpretations there-of) used in the simulation experiments. The qualitative properties of the

[25] It is also the reason for the utility of evolutionary camouflage capabilities in organisms such as chameleons and arctic foxes.



(a) Scenario R4

(b) Scenario R3

Figure 6.30: Images captured by the onboard camera during an execution of scenario R3 (task 2). In each case, the obstacle appears on the right. The wooden block significantly blends in with the background colors of the table, while the metal bar has a higher contrast to the background

trajectories were largely similar, leading to similar conclusions that are briefly discussed in the following.

From the batches of task executions in each scenario, for which we maximize the consistency of task conditions, we determined an overall success rate of 87.5% by averaging across scenarios. This metric indicates a moderately high level of reliability, even though a corner case involving a problematic obstacle texture (R4) was included in the experiments. The effect of this case in the overall estimation of performance is less pronounced in the median overall success rate, which is approximately 91.7%. To the extent of the coverage of these experiments, the success rates in individual scenarios indicate that the implementation is:

- similarly reliable in two different backgrounds
- as reliable in relatively dim lighting conditions as in significantly brighter conditions
- less reliable the more an obstacle's texture/color blends with the background's texture/color

For the assessment of predictability, we estimated $\dot{\zeta}$, removed outliers, and plotted the values in histograms and density estimate curves as before. A plot for scenario R2 is displayed in Figure 6.31. Once again, we note that the distribution of $\dot{\zeta}$ spans low values which indicate that the end-effector does not exhibit sudden and large changes in direction along its trajectory. Compared to the simulation results, the case in which the obstacle avoidance module is active shows slightly larger $\dot{\zeta}$ values than when it is not, though these differences seem insignificant with respect to the absolute magnitudes. These small differences may again be attributable to the changes in controller parameters and/or discrepancies in actuation effects in the simulation. The SNN obstacle avoidance module can therefore be said to generate predictable motions on the real arm as well; as in the simulation; in other words, subsequent trajectory positions can be predicted with relative ease by observing the preceding portion of the trajectory. This conclusion has been verified through visual observations of the arm's executions.

Along the dimension of safety, the executed trajectories on the real robot were on average fairly safe, but to a lesser degree than in the simulation. Again, this was concluded by examining the distributions of instantaneous velocities applied at the end-effector (Figure 6.29a) and drawing comparisons to a reference safety threshold of $0.25m/s$ obtained from the ISO standard for industrial robot safety requirements (ISO 10218-1:2011 (2011)). While the overall (normed) speed of the end-effector had a mean value of approximately $0.13m/s$, thus remaining within the safety bound, some values exceeded the threshold,

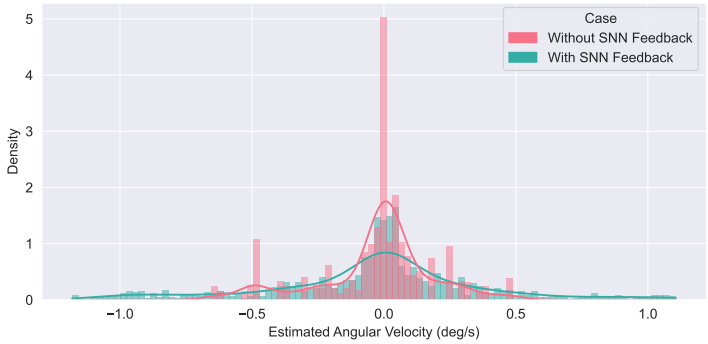


Figure 6.31: A histogram and kernel density estimate plot depicting the distributions of estimated angular velocities of the end-effector measured during real robot experiments in scenario R2.

with the task 1 and task 2 maximums approaching $0.5m/s$ and $0.55m/s$, respectively. These maximum values were higher than in the simulation experiments by $\sim 0.2m/s$, probably due to the aforementioned modification of controller gains. However, the speeds exceeding the threshold lie beyond the third quartile (75th percentile) of the distribution (upper plot), and represent a relatively small region of the density (lower plot)[26]. As with the reliability measure, it is reasonable to assume that this drop in safety is not an effect of the obstacle avoidance maneuvers themselves, but rather the overall controller parameterization (given that similar values are observed in the "Without SNN Feedback" case). The perceived safety of the arm's motions could be improved through more tuning of control parameters.

With regards to the naturalness criterion, we can observe from an initial examination of the trajectory shapes, depicted in Figure 6.28, that they do not appear uneven or erratic, similar to the trajectories observed in simulations[27]. For a more objective measure of smoothness, we applied the same jerk analysis procedure (refer to section 6.1.4 for an explanation). Figure 6.32 depicts a sample trajectory from scenario R2 executions for which the jerk profile was analyzed. Figures 6.33–6.35 show the plots of positions, estimated jerks, and estimated jerk densities for trajectory segments and their analogous theoretically minimum-jerk trajectory segments in x, y, and z. As in the simulation executions, the arm produces jerks whose average is expectedly higher than the theoretical minimum and which have a larger variance (specifically in y and z), but the difference is not large. This verifies the conclusions on smoothness that were drawn from visual observations: the executed trajectories are not optimally smooth, but do not contain noticeably sudden accelerations or unusual transitions between positions that would characterize the arm's motions as unnatural.

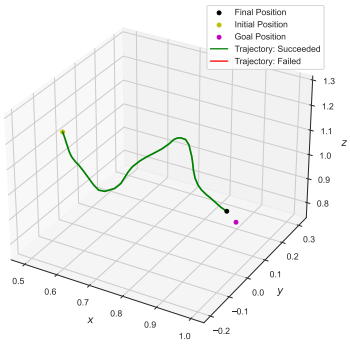


Figure 6.32: Sample trajectory (scenario R2)

[26] In other words, the estimated probability of observing values exceeding 0.25, which could be obtained by computing the area under the density curve in the region >0.25, would not be large.

[27] Unlike in the simulation results, the 3D trajectory plots do not include the positions of the obstacles, which were readily available from the simulation.

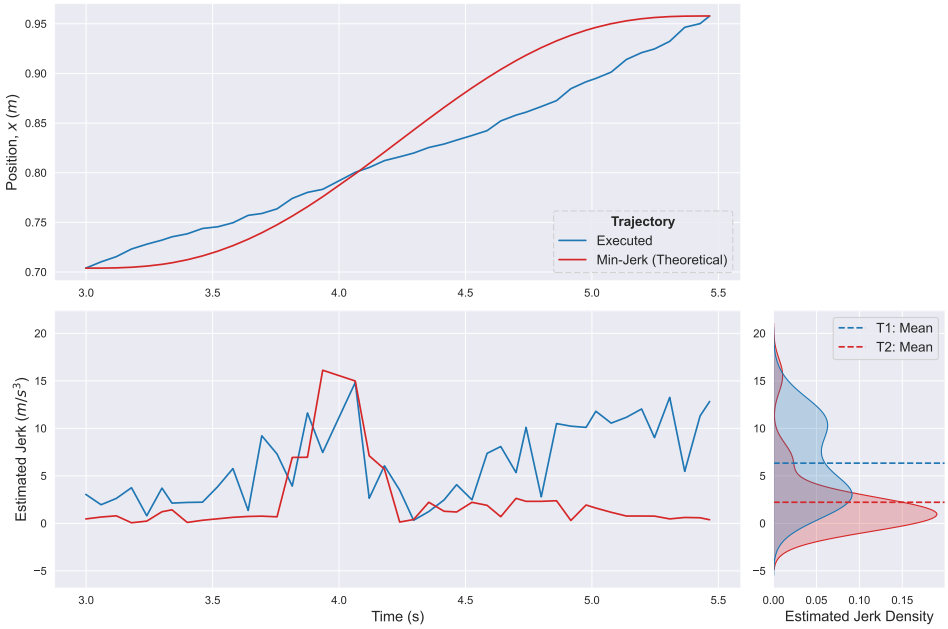


Figure 6.33: A comparison of estimated jerk between a section of the sample trajectory (Figure 6.32) in x (between 3.0 and 5.5 seconds in the execution) and a minimum-jerk trajectory. The plot contains the positions (top), jerks (bottom left), and estimated densities of jerk values (bottom right).

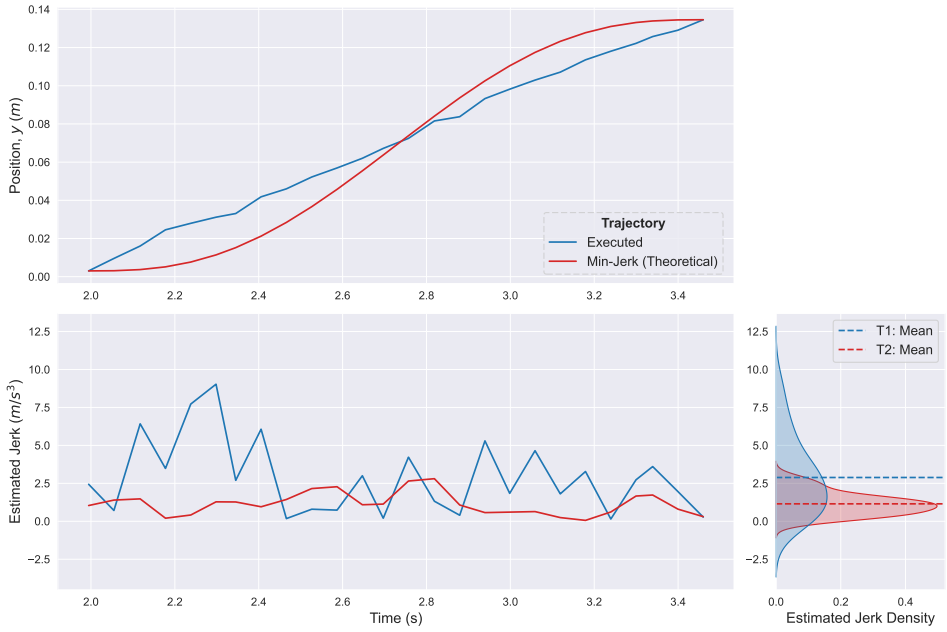


Figure 6.34: A comparison of estimated jerk between a section of the sample trajectory (Figure 6.32) in y (between 2.0 and 3.5 seconds in the execution) and a minimum-jerk trajectory. The plot contains the positions (top), jerks (bottom left), and estimated densities of jerk values (bottom right).

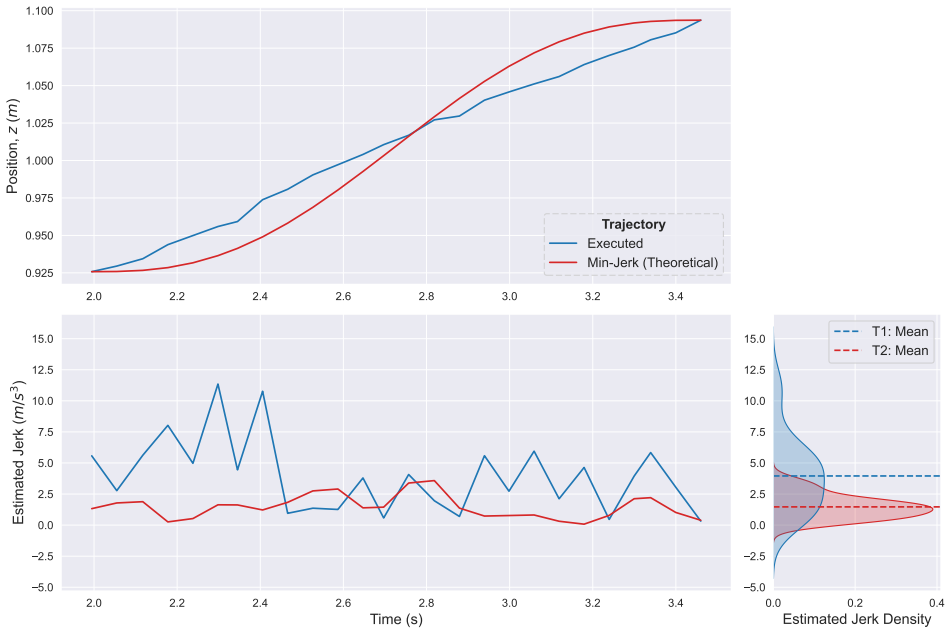


Figure 6.35: A comparison of estimated jerk between a section of the sample trajectory (Figure 6.32) in z (between 2.0 and 3.5 seconds in the execution) and a minimum-jerk trajectory. The plot contains the positions (top), jerks (bottom left), and estimated densities of jerk values (bottom right).

6.3 Experiment Conclusions

Concluding this discussion of experiment results in simulation and on the real robot, we now summarize the most relevant outcomes, notable insights from both sets of experiments, and some observed limitations as well as possible improvements to the experiments we designed and executed.

The current implementation of our proposed approach achieved frequent success in avoiding static and dynamic obstacles, and the conducted experimental trials provided statistically significant results confirming and indicating the extent to which it outperformed non-adaptive nominal executions. Within the set of designed tasks and the wide range of task conditions, the SNN-based obstacle avoidance module demonstrated median success rates of 84% and 92% in simulated and real experiments, respectively, involving imminent collision situations. Through the defined quantitative metrics, we analyzed the performance of various parameter combinations in order to iteratively tune and refine our implementation. In the final experiments, execution times, trajectory lengths, and velocity and acceleration magnitudes indicated that adaptive, obstacle-aware trajectories were quantitatively similar to nominal executions, but with the added capability of consistent obstacle avoidance. Furthermore, a detailed qualitative assessment suggested that the adaptive trajectories were moderately reliable, usually predictable, adequately safe, and likely smooth enough not to be perceived as unnatural, although they are only directly optimized for obstacle avoidance.

The initial experiments conducted in simulation yielded a variety of practical insights. Although success was fairly consistent in simpler scenarios, some task conditions proved to be more challenging,

such as the cluttered, dimly-lit "Office" background. This, in addition to minor differences for different obstacle types/shapes, could be addressed through further parameter tuning. Results in task 2 showed that performance transferred relatively well to a novel task but exhibited some degradation compared to tasks that were observed and tuned for, indicating that the proposed approach still relies on good parameter tuning. Through the occasional unfortunate evolutions of some trajectories executed in task 3, which lead to situations in which it was difficult to avoid the obstacle in time, we found a perceptible level of uncertainty in trajectory outcomes due to the complex interactions of components and cascaded operations within the pipeline. The degree of this uncertainty and sensitivity to subtle variations in input stimuli would be interesting to quantify and study, but is left for future work involving deeper analyses. In task 4, high-speed obstacles proved to be a limiting corner case for the current implementation and the chosen parameterization, such that increasing the sensitivity and thus reactivity to fast obstacles introduces undesirable safety implications. In spite of this fundamental limitation, the relatively higher measurements of instantaneous velocities and accelerations in these scenarios have shown that the magnitude of the reaction seems to correlate with the obstacle's speed. In other words, a higher speed obstacle seems to induce larger neural activations which translate to higher applied accelerations, demonstrating intuitively proportional responses from the SNN-based obstacle avoidance module.

An integral aspect of our approach to development and experimentation was the tuning $\rightarrow$ validation $\rightarrow$ testing procedure. With a systematic method of testing on a variety of structured task environments, we were able to draw the aforementioned conclusions and various interesting characteristics of executed trajectories from repeated trials with reasonable degrees of certainty. More importantly, the ML-inspired approach helped in identifying difficulties in particular scenarios and addressing them by tuning parameters. As an example, parameter sets 10-12 were successfully tuned to mitigate issues observed in earlier sets in task 4, significantly improving performance in that task and task 3. The ability to accomplish this is also evidence that the parameters of our pipeline are adequately interpretable, such that we could identify and predict the effects of each and then tune them to achieve the desired modifications of behaviour.

The computation time analysis showed that the SNN computation time depended on the amount of motion in the image, which validates two concepts that are central to our application of SNNs in this project. Firstly, this verifies the property of only processing relevant information in SNNs, which has the potential of saving energy and time, particularly when compared to conventional DNN image processing, where all RGB values are constantly processed even for a completely stationary scene. Secondly, this highlights the homogeneity of EC data and SNNs postulated at the beginning of this study; since the relevant information for the problem we address (and perhaps many others) is the relative change in the input data, events efficiently distill this information and the SNN selectively performs computations only on those portions of the inputs represented by the events and, by extension, derived spikes. This is possible due to the established property of the SNNs, but which is only fully realized with the kind of data provided by ECs.

Among the primary outcomes of real robot experiments is a validation of the transferability of our approach and implementation to a real setting. In particular, we were able to achieve comparable performance in addition to quantitatively and qualitatively matching behaviour on the real robot, only

requiring slight, intuitive adaptations to tuned parameter values. These adaptations amounted to controller gains and the event camera emulation threshold, which were necessary due to discrepancies between actuation effects and the quality of RGB camera images in the simulation in comparison to the real setting. While we may expect the results of simulation experiments to exhibit some variance due to simulation latency effects if run on different machines, the real experiments serve to diminish this source of uncertainty. Experiment results verified that our neuromorphic approach to obstacle avoidance achieved fairly reliable behaviours in imminent collision scenarios that necessitated real-time processing and reactivity.

As we demonstrated during the parameter tuning phase, our implementation is conducive to extensions that enforce desirable task constraints, such as the positional workspace limits and the safety strategies. Similarly, the implementation could be improved further with novel safety strategies and features such as velocity reductions in specified conditions, should the need arise.

6.4 A Comparison of Event Emulation Strategies

We conducted an analysis of the results of different event emulation methods and their effects on the performance of our obstacle avoidance pipeline. This analysis involved investigating differences in the derived events, their effects on SNN activations, and the resulting obstacle avoidance performance.

As described in section 4.1.2, our default event emulation strategy involves computing the differences in absolute intensities at every pixel and emitting an event wherever this quantity exceeds θ. This is repeated in the following equations, which describe the intensity difference calculation (6.12), and the polarity of event an e_k that is emitted when either condition is fulfilled (6.13):

$$\Delta \mathbf{L}_{abs}(\mathbf{x}_k, t_k) = \mathbf{L}(\mathbf{x}_k, t_k) - \mathbf{L}(\mathbf{x}_k, t_{k-1}) \tag{6.12}$$

$$p_k = \begin{cases} +1, & \text{if } \Delta \mathbf{L}_{abs,c}(\mathbf{x}_k, t_k) > \theta, \, \forall c \in \{R, G, B\} \\ -1, & \text{if } \Delta \mathbf{L}_{abs,c}(\mathbf{x}_k, t_k) < -\theta, \, \forall c \in \{R, G, B\} \end{cases} \tag{6.13}$$

Here, we make explicit the condition that the intensity difference must exceed θ in **all** three color channels to trigger an event, which we term a "multi-channel" condition. This method is identified as M1.

The second method (M2) differs from the first in blurring the source RGB images before computing the same absolute intensity differences. This is similarly applied in Zahra et al. (2021) for the purpose of removing high-frequency noise; here, we aim to investigate its effects particularly in derived events. We apply blur by convolving an image with a low-pass filter represented by a normalized 5×5 box kernel, i.e. an array of 1's normalized by $\frac{1}{25}$[28].

In M3, we modify the conditions of M1 to compute the differences in *log* intensities:

$$\Delta \mathbf{L}_{log}(\mathbf{x}_k, t_k) = log(\mathbf{L}(\mathbf{x}_k, t_k)) - log(\mathbf{L}(\mathbf{x}_k, t_{k-1})) \tag{6.14}$$

This mimics how most real event cameras measure irradiance changes, resulting in their characteristically high dynamic ranges (Rebecq et al. (2019), Gallego et al. (2022)), and is often used in event data emulation

[28] We utilize OpenCV's existing *blur* function.

(Rebecq et al. (2018)). The multi-channel event emission condition of Equation 6.13 remains the same, except for a necessary change in the θ value to adapt to the difference in magnitudes.

The fourth method (M4) is based on an emulation strategy employed in Salvatore et al. (2020), termed the "Salvatore" method, where the log of a sum of weighted channel intensity values is used to compare the difference:

$$\begin{aligned}\Delta L(\mathbf{x}_k, t_k) = \; & log(0.299L_R(\mathbf{x}_k, t_k) + 0.587L_G(\mathbf{x}_k, t_k) + 0.114L_B(\mathbf{x}_k, t_k)) \\ & - log(0.299L_R(\mathbf{x}_k, t_{k-1}) + 0.587L_G(\mathbf{x}_k, t_{k-1}) + 0.114L_B(\mathbf{x}_k, t_{k-1}))\end{aligned} \tag{6.15}$$

Note that L is in this case a scalar and the event emission condition is modified accordingly:

$$p_k = \begin{cases} +1, & \text{if } \Delta L(\mathbf{x}_k, t_k) > \theta \\ -1, & \text{if } \Delta L(\mathbf{x}_k, t_k) < -\theta \end{cases} \tag{6.16}$$

Finally, method M5 mimics the emulation strategy implemented in the open-source *pyDVS* emulator (García et al. (2016)), which encodes pixel intensities using Gamma functions, instead of using absolute or log values, and incorporates additional operations that refine the resulting event data. We modified the open-source implementation[29] to integrate and run within our pipeline as a substitute to the `event_image_streamer` ROS node in a dedicated `pydvs_event_image_streamer` node.

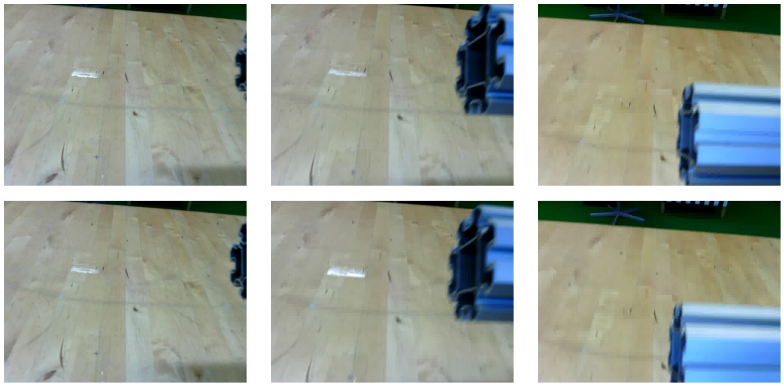
In summary, we considered the following event camera emulation methods in our analysis:

- **M1:** Multi-channel RGB absolute differences
- **M2:** Multi-channel RGB absolute differences, with blur
- **M3:** Multi-channel RGB log differences
- **M4:** "Salvatore" method
- **M5:** "pydvs" method

While all methods can be run interchangeably within the pipeline, it was necessary to adjust the θ value in some cases to produce sensible outputs. These adjusted values were empirically determined through experimentation and observations of the resultant event data. For M1 and M2, we use the θ value from the final parameter set in our experiments (set 12): 28. For the log-based difference measure of M3, we set $\theta_{M3} = 1.0$, which is more suitable for the significantly smaller range of values spanned by the differences in log-intensities in comparison to absolute intensities. For M4 and M5, we set the threshold values to $\theta_{M4} = 0.5$ and $\theta_{M5} = 70$, respectively.

Firstly, we compared the visual qualities of the event data generated by each method and the outputs of the SNN in response to each. To that end, we extracted sample images from a scenario R3 trial executed during the real robot experiments to emulate event data from. In particular, we sampled three pairs of consecutive images, shown in Figure 6.36. The visual representations of the event data ("event images") produced by each method are illustrated in Figure 6.38. These data were used as inputs to the SNN, from

[29] https://github.com/chanokin/pyDVS



(a) Sample 1 (b) Sample 2 (c) Sample 3

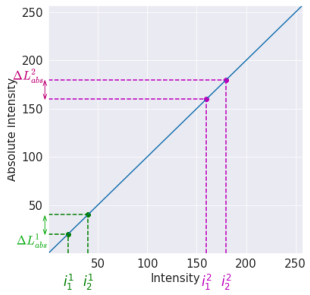
Figure 6.36: Source images sampled from a scenario R3 execution on the real robot. Each column shows a pair of consecutive images from which event images are derived.

which we recorded the spike trains output in a period of 40ms (simulation time, T_{sim}) as a result of each input. In Figure 6.39, each sub-plot depicts the recorded spike trains (left) and the total count of spikes (right) across T_{sim} at each output neuron. Note that the shared y-axes contain the indices of the flattened array of output neurons.

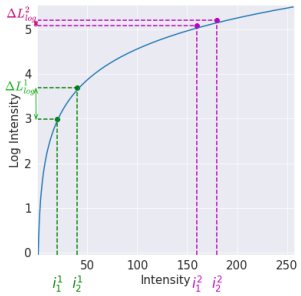
Figure 6.38 shows subtle variations in the emulated event data. We observe that the blurring applied in M2 leads to mostly similar event images but additionally eliminates some spurious events, seemingly caused by noise or insignificant intensity changes, such as in the top regions of samples 1 and 2. This indicates that blurring source images could have an effect of retaining the most significant events while reducing undesirable noise.

As expected, the distributions of ON and OFF events appear notably different in M3. For instance, darker regions, such as in the interior of the metal bar, produce significantly more events than brighter regions, such as on the outer surface. This is likely a result of the higher sensitivity of the log-based difference measure for darker colors; since the slope of the log function is significantly steeper at lower values, two lower intensities produce a larger ΔL_{log} than two higher intensities that are equally different (unlike the default, linear measure, ΔL_{abs}), and are therefore more likely to cross the emission threshold (see Figure 6.37 for an illustration).

The fourth method, which also compares intensities in the log space but through a specifically weighted sum of RGB values, tends to produce events with a lower contrast between dark and bright regions. The resulting event data is denser in the most relevant regions (containing obstacle motion), as it seems to adequately combine the bright region events from M1 and M2 and the dark region events from M3. Results from the Gamma-based measure of



(a) Absolute Intensities



(b) Log Intensities

Figure 6.37: $\Delta L^1_{log} > \Delta L^2_{log}$, since i^1 intensities have lower absolute values (6.37b), whereas $\Delta L^1_{abs} = \Delta L^2_{abs}$ (6.37a).

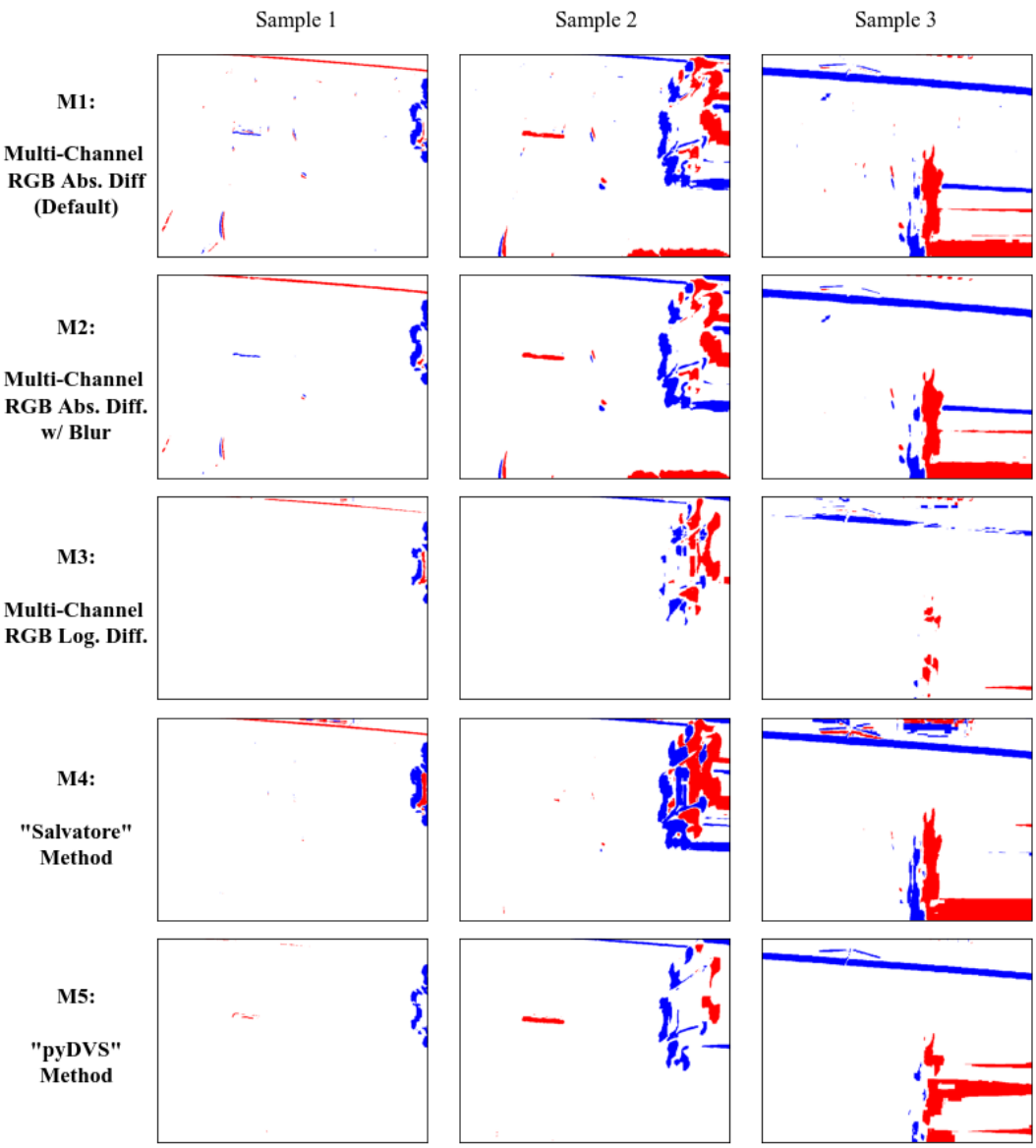


Figure 6.38: Event images produced by each emulation strategy for each of the three sample source shown in Figure 6.36

M5 look fairly similar to those of M2 except for a slight reduction in apparent noise (in this case, this may be more attributable to the selected θ values rather than the two methods themselves). Similar to the first two methods, M5 is susceptible to reacting strongly to reflective surfaces, such as the one present near the center of the image, which produce high and unstable intensities; the log-based measures do not suffer from this problem.

Interestingly, the SNN responses visualized in Figure 6.39 seem to reflect the input events, confirming that the output activation map is essentially a spatio-temporally-filtered, down-scaled version of the input

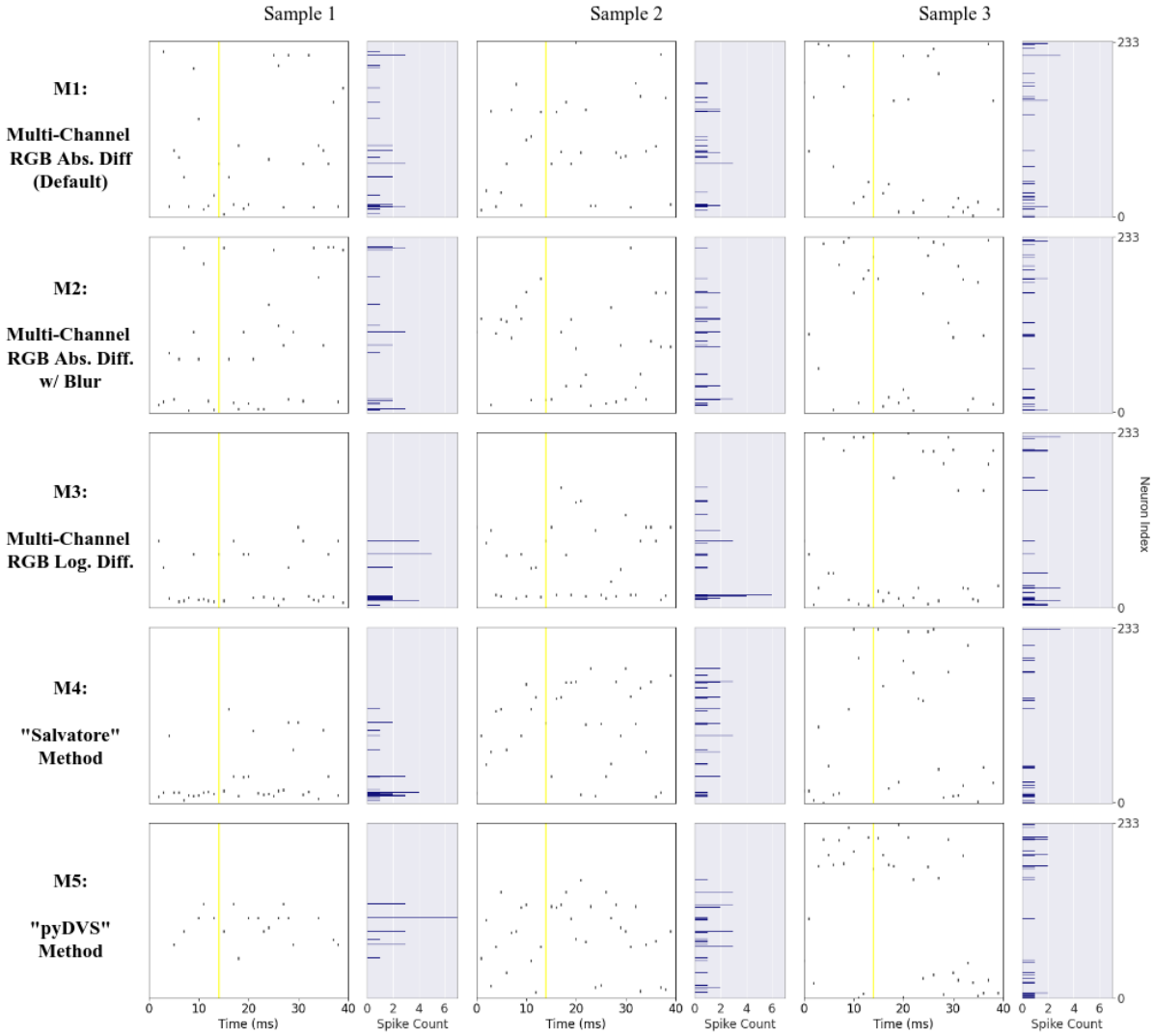


Figure 6.39: SNN responses to input events from each emulation strategy for each sample. The plots show the spike trains (left) and spike counts (right) for each output neuron.

events representation, as mentioned in section 4.1.4. Note that, as depicted in Figure 6.40, the spikes plots are indexed by a row-major re-ordering of the indices of the output neurons, which are originally arranged in a two-dimensional grid, where the first and last indices are in the top-left and right-bottom positions, respectively.

For sample 1, the first four methods have significant spike counts in the first few neurons (near the bottom), which correspond to upper regions of the image. Indeed, the events visualized in Figure 6.38 show that they all have strong event activity at the top, except for M5. M4 and M5 have almost no spiking activity in neurons corresponding to lower regions of the image, except for sample 3, which is similarly apparent from their event images. Although the blurring strategy of M2 produces less noisy events than M1, this distinction is not apparent from the spike plots, which show very similar spike distributions. A likely reason is that the SNN's dynamics would filter out spurious inputs anyway, which explains the very similar responses and indicates that the SNN could obviate the need for such a denoising operation. The different effects of the log-based methods (M3 and M4) can be observed in their similar spike trains and distributions, in comparison to M1 and M2, which is especially apparent in sample 1. Generally, the SNN's responses are similar across methods, showing a degree of robustness to variations in events (see, for example, the spike trains concentrated in the top and bottom regions of the plots for sample 3 across methods). Only the responses to M5 input data seem to slightly dissimilar to the others. Indeed, the ultimate obstacle avoidance response would also be similar due to our first-spike-time decoding strategy (refer to section 4.1.4): we consider neurons that fire before t_{act} (depicted as a yellow line in the Figure) "active" and indicative of "obstacle points", and neurons that spike first are generally around similar positions between methods.

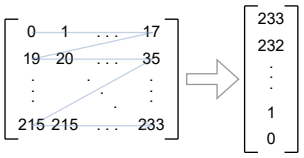


Figure 6.40: An illustration of the row-major re-ordering of output neuron indices defining the vertical axes of Figure 6.39.

Next, we repeated N_{trials} executions of one of the testing set scenarios (30) for each method in order to compare the resultant obstacle avoidance performance through our quantitative metrics. Figure 6.41 contains a plot of the results.

Evidently, the results of the SNN-based obstacle avoidance module do not significantly change between the presented methods. The largest difference in results is observed with M5, whose success rate of 85% deviates the most from the mean (94%). These results verify that the ultimate obstacle avoidance behaviours are fairly similar, as we would expect from the similarity in SNN responses to each method.

In this analysis, we have examined the properties of a selection of event emulation methods and their effects on our approach. Though we have concentrated on this set, other methods could be explored in future analyses, such as operating in the HSV or grayscale color spaces, single-channel methods, etc. The spiking responses and obstacle avoidance performance we showed indicate that our SNN is robust to differences in the event data from the tested methods, since they inherently act as spatio-temporal filters that are tolerant to some noise or variance. A different SNN output decoding strategy that, for example, considers more than neurons' first spike times may result in more nuanced and varied responses and behaviors between the EC emulation methods and is thus worth exploring in future work. Nevertheless,

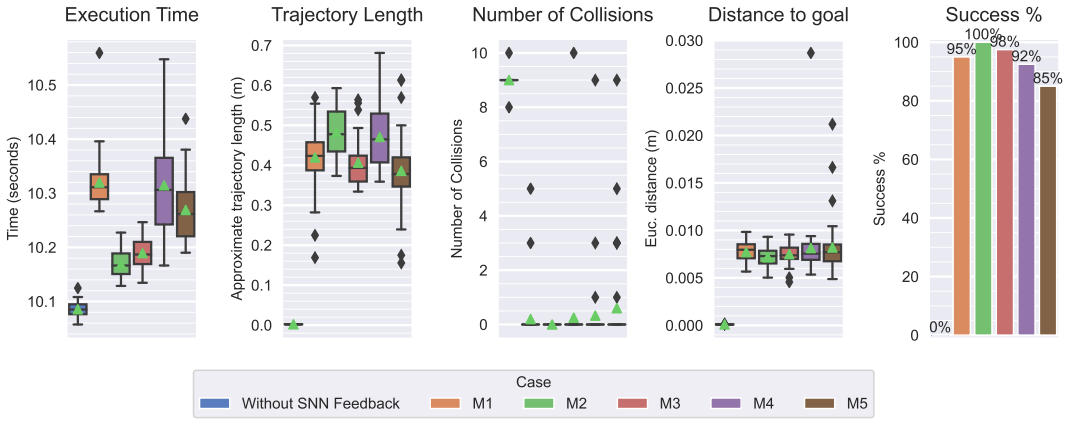


Figure 6.41: Quantitative metric results for testing scenario 30: without SNN feedback and four batches with parameter set 12, each with a different event emulation method: M1 (default) to M5.

our basic implementation of event emulation currently performs as well as others from the literature (such as M4 and M5) for our problem, in which some inexactness in spike locations is acceptable. Finally, it is worth noting that the results presented here are partially dependent on the selected values of θ, and further independent tuning for each method may provide more insights.

6.5 Decoding Avoidance Behaviour From Raw Event Data

Previous results, such as of the analysis presented in the last section, have provided validations of the merits and utility of the SNN in our neuromorphic approach. In an additional experiment, we further investigated this utility by completely removing the SNN component from the pipeline and studying the resulting obstacle avoidance behaviour. To achieve this, we decoded obstacle avoidance trajectory adaptations directly from raw events, instead of the spikes output by the SNN in response to events. Further, we also evaluated performance resulting from decoding completely random events, in order to verify that the presented results are indeed due to the information in the event data and subsequent SNN processing.

The decoding of raw events into obstacle avoidance velocities was possible due to the homogeneity of event and spike data. In the original implementation, the first-spike-times of output neurons, particularly those that do not exceed t_{act}, define the neural activation map which contains the "obstacle points" in the output feature space. Instead, we resize the event images output by the `event_image_streamer` to match the SNN's output feature space[30], and designate events within this space as the "obstacle points". The potential fields-based decoding procedure that follows (described in section 4.1.4) is otherwise identical, resulting in the $\boldsymbol{\phi}$ accelerations that in turn dictate trajectory adaptations. In the pipeline, we substituted the `snn_simulator` node with an `event_output_generator` node, which handles the transformation of the event data into the same format of the SNN output which can be processed by the `motion_controller`.

[30]The resizing is done using bilinear interpolation, which is available in the OpenCV software package.

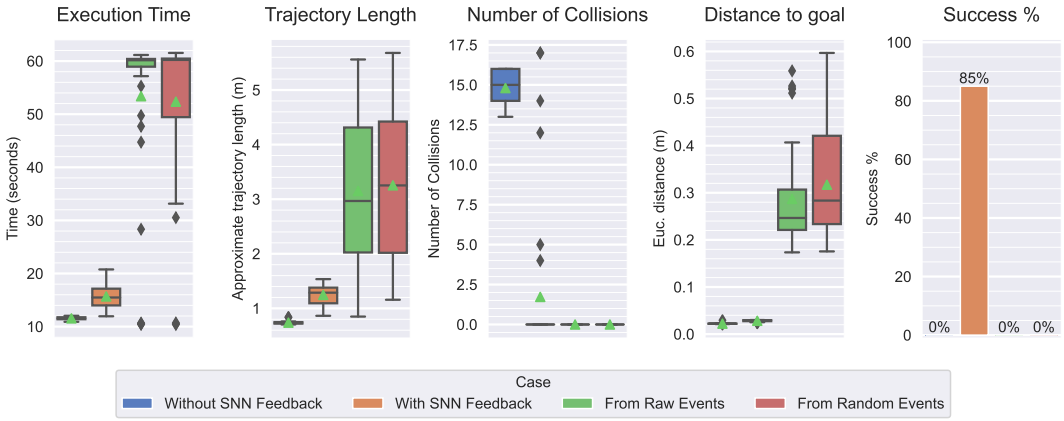
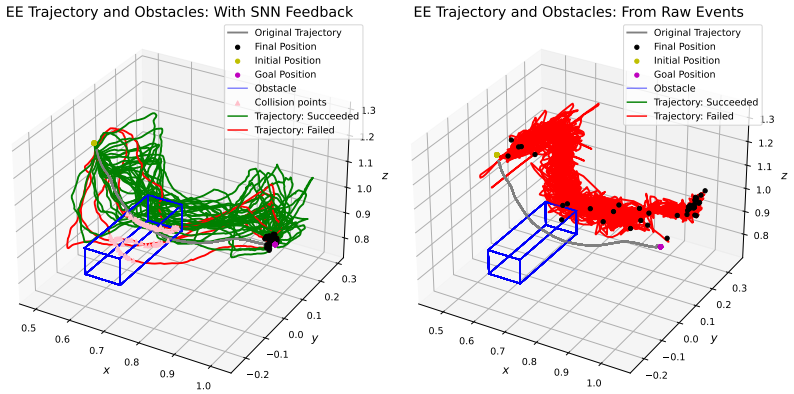


Figure 6.42: Quantitative metric results for testing scenario 25: without SNN feedback, with SNN Feedback (parameter set 12), and decoding obstacle avoidance behaviour from raw or random events. Note that all except the second batch have zero success.



(a) From SNN neural activations (b) From raw events

Figure 6.43: A comparison of trajectories executed in simulation testing scenario 25 between decoding SNN outputs (default) and raw event data.

We selected scenario 25 from the testing set of the simulation experiments to quantitatively evaluate the effects on obstacle avoidance performance. As before, we ran $N_{trials} = 40$ executions, this time with i) decoding from raw events and ii) decoding from random events.

In both cases, the robot failed in all executions of the task, despite subsequent efforts to tune parameters for stable obstacle avoidance accelerations. We illustrate the metric results for each case in Figure 6.42 and show the trajectories executed when decoding from raw events in comparison to the SNN outputs in Figure 6.43. During these trials, the arm would oscillate due to the accelerations induced by the raw events and ultimately never reach the goal.

These results provide further evidence of the importance of the SNN within our neuromorphic obstacle

avoidance approach. More specifically, the inadequacy of raw event data for producing successful obstacle avoidance trajectory adaptations indicate that the neural dynamics of the SNN are integral for processing that data to achieve the desired outcome. Furthermore, the similar negative results obtained from totally random events or the raw events additionally confirm the insufficiency of just the event data in our approach and validates the utility of the SNN.

6.6 Random SNN Weight Initializations

In all tests discussed in this chapter, we constrained the SNN weights to a set of random but fixed values in order to draw conclusions on the effects of various parameter values and methods from statistical comparisons of the results by eliminating variance due to the synaptic weights. In this section, we present results of an analysis of performance with different random weight initializations, with the aim of investigating the dependence on SNN weight values.

As in the previous analyses, we selected a scenario from the testing set (31) and ran eight batches of N_{trials} trials in simulation, parameterizing the SNN with a different set of random weights in each batch. In each case, the weights are sampled from the same uniform distribution (see Equation 4.3 in section 4.1.3) but with a different random seed. For this analysis, we run a larger number of $N_{trials} = 60$ trials per batch, in order to further minimize variances.

Figure 6.44 shows the metrics plot for these runs (note that "Seed 1" refers to the batch executed during the original experiments). The rate of success was found to somewhat vary around a mean of 90.5% with a standard deviation of 3.5%. This indicates that the weight values have a non-negligible, albeit not excessive, effect on obstacle avoidance performance. With the exception of occasional outliers, the distributions of the other metrics are mostly similar across the weight initializations. Besides statistical comparisons, no differences between the trajectories produced in each case were perceptible from visual observations of the trials.

Therefore, we observe some variation in the ultimate results of the SNN-based obstacle avoidance

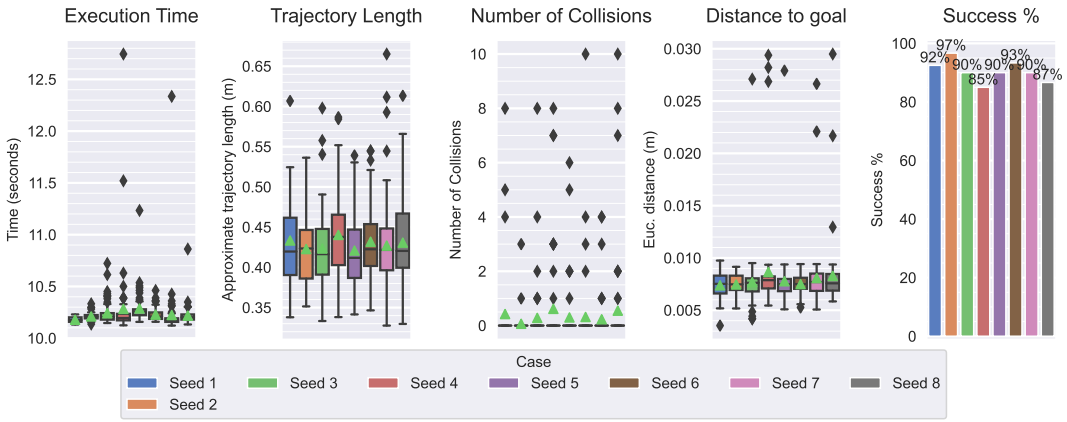


Figure 6.44: Quantitative metric results for testing scenario 31 with parameter set 12, repeated with eight random (but fixed) SNN weight initializations.

module with different sets of randomly-sampled weights, which shows a degree of sensitivity to SNN weight values. This variation in results is not significant enough to invalidate previously established conclusions and, in fact, has positive implications for future applications of learning. In particular, the observed relevance of the weight values suggests the possibility of searching for a set of values that improve obstacle avoidance performance. This insight motivates the application of optimization and learning algorithms to tune SNN weights in the pursuit of further improving obstacle avoidance performance; an interesting avenue for future work.

6.7 Real Event Camera Tests

For this thesis project, we developed and utilized event emulation within our neuromorphic pipeline as a substitute for a real EC, aiming to draw conclusions about the feasibility and effectiveness of the neuromorphic approach, with the hypothesis that the conclusions we arrive at can be extended to a system equipped with an EC. In order to facilitate incorporating ECs in future extensions, the emulated event representation and the pipeline components were intentionally designed such that emulated or real event data can be processed interchangeably. As an initial step towards this objective, we integrated an EC within our pipeline and conducted preliminary tests on the real robot. In this section, we present and discuss the results of these tests.

Figure 6.45: The iniVation DAVIS346

We used the DAVIS346 EC[31], pictured in Figure 6.45, which contains a dynamic vision sensor (DVS) and active pixel sensor (APS) that enable capturing event and conventional RGB data, respectively, in addition to an IMU. The DAVIS346 has a relatively large pixel size of $18.5\mu m^2$ and a correspondingly small resolution of 346x260 [32]. By comparison, the Omnivision OV5640 color sensor mounted on the Kinova arm has a pixel size of $1.4\mu m^2$ and a resolution that ranges from 320x240 to 2592x1944 (with a decreasing frame rate). However, the DAVIS has a significantly higher dynamic range than the OV5640 ($120dB$ in contrast to $68dB$), and thus supports a wider range of illumination levels. In addition, the OV5640 consumes $\sim 700mW$ while the DAVIS consumes only 10-30mW to transmit event data and an additional $140mW$ if the APS is active, for a 5V DC supply[33]. The DAVIS was attached to the top of the end-effector to run our tests.

Firstly, we examined the event images obtained from the DAVIS's event streams in comparison to those we emulate. We recorded the EC's events and color frames for a simple hand motion, then used the color frames as inputs to our emulator to generate emulated event data from approximately the same visual input. Figure 6.46 depicts the events generated for four frames of the motion by the emulator and the DAVIS. Evidently, the emulated events are spatio-temporally similar to those of the DAVIS, but the latter is more sensitive to minute motions and produces significantly more salient events. Note, for

[31] The DAVIS346 is supplied by iniVation.

[32] This is characteristic of current event cameras, which significantly lag behind conventional cameras in pixel size and resolution. See Gallego et al. (2022).

[33] The OV5640's power consumption was derived from reported consumption in mA and usual voltage supply rating, since a figure in mW was not provided in the specifications.

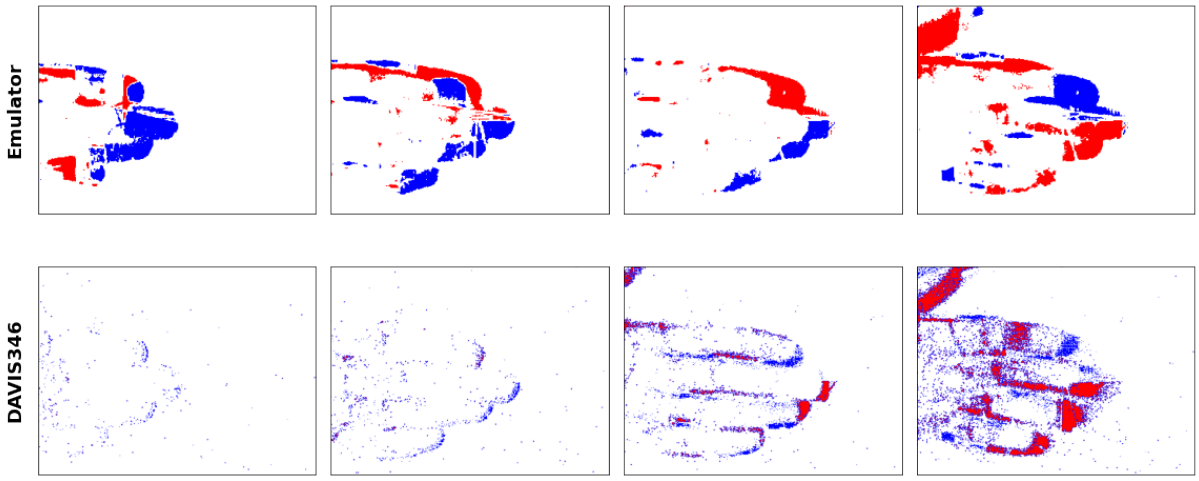


Figure 6.46: A comparison of events generated by our emulator and the DAVIS346. Note RGB images captued by the DAVIS346 were used to generate the emulated event images.

example, the DAVIS's superior ability to capture events at borders in the image. On the other hand, the DAVIS's output is notably noisier; this is commonly observed in EC data (Milde et al. (2017)) but may be alleviated with careful tuning of the camera's parameters (just as more tuning of our emulator may refine the outcome). It is important to note that, with the default settings, the color frames obtained from the DAVIS appear to have lower color quality and contrasts and are captured at a relatively low rate compared to the OV5640 ($\sim 12Hz$ as opposed to $\sim 30Hz$). As would be expected, these factors have an effect on the emulation efficacy and seem to contribute to the comparatively lower sparsity of the emulated events and the blurring effect which had not been observed in previous event images (see Figure 6.38).

For an evaluation of the DAVIS's performance within our SNN-based obstacle avoidance module, we incorporated the camera instead of the emulator and repeated executions of scenario R3 of the previous experiments. We utilized the functionalities implemented in the open-source *rpg_dvs_ros*[34] for configuring the camera and publishing the data in ROS topics. An additional `dvs_events_data_converter` node was developed to transform the event data, which is transmitted in an array of tuples, to the same events image representation produced by the `event_image_streamer` node, which it replaces.

The DAVIS can be configured through a relatively large set of hardware and software parameters, including voltage thresholds, exposure gains, special noise removal options, and emission thresholds for ON and OFF events (resembling our θ parameter). These parameters, particularly the emission thresholds, were moderately adjusted to induce SNN responses of similar magnitudes to those induced by the emulator. However, the DAVIS was found to still produce denser and noisier event data on average,

[34] https://github.com/uzh-rpg/rpg_dvs_ros

which necessitated diminishing the sensitivity to the event data elsewhere in the pipeline. To that end, we modified the values of two parameters (of parameter set 12): ϕ_{max} from 4000 to 2000, and s_{BE} from 5 to 3. The former reduces the limit on instantaneous avoidance accelerations computed from the PF, thus damping avoidance velocities, while the latter reduces the effects of binary erosion filtering (which lead to better results following the change to ϕ_{max}).

Due to the Kinova arm's design, the DAVIS had to be placed atop the integrated camera module, resulting in a vertical displacement in the FOV with respect to the original scenario R3 executions. The scenario specification, including the obstacle type (metal bar) and its trajectory, was adhered to except for slightly increasing the height at which the obstacle traveled (i.e. z coordinate) in order to adapt to this discrepancy. As in the original experiments, $N_{trials} = 30$ were executed and then compared to the previous results.

Figure 6.47 illustrates the quantitative results of scenario R3 with event emulation (presented in section 6.2 and with the DAVIS346 camera. Most significantly, we observe a similarly high rate of success with the event camera in this scenario, despite three observed failures. As in the original experiments, even light touches of the obstacle following its avoidance are considered failures; this was the case in two of these instances. Figure 6.48 shows the trajectories executed in each case, which mostly appear similar, though the EC executions seemed more inhibited. Ultimately, the minimal tuning of the DAVIS's parameters lead to generally similar results.

In this section, we have shown through a preliminary analysis that our obstacle avoidance pipeline supports the seamless substitution of the event emulator with an EC. In addition, we achieved reasonably similar task performance in the tested scenario following minimal parameter adjustments, which was verified through executions on a Kinova arm equipped with a DAVIS346 camera. These results, in addition to the relative similarity in output event data, indicate that conclusions arrived at in this chapter could be extended to a system incorporating a real EC, and provides some validation for using event camera

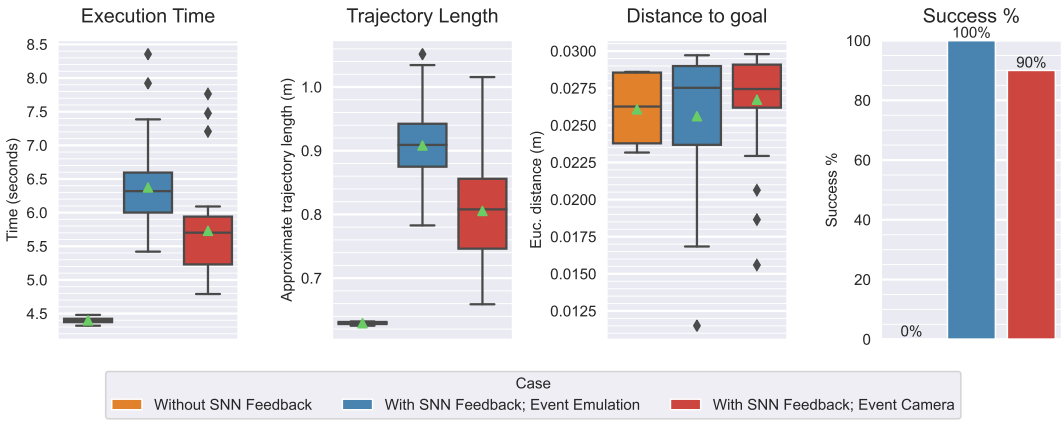
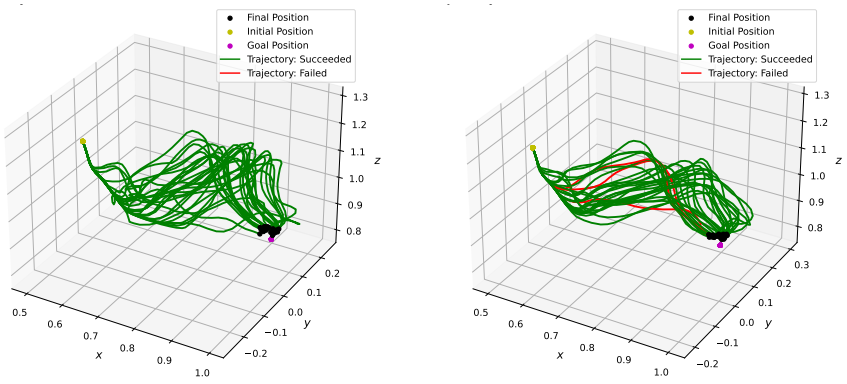


Figure 6.47: Quantitative metric results for real robot testing scenario R3 with parameter set 12 with results when using the event emulation (obtained during the experiments discussed in section 6.2) and the real event camera.



(a) With the event camera emulator (b) With the DAVIS346 event camera

Figure 6.48: A comparison of trajectories executed on the robot in scenario R3 with the event camera emulator and the DAVIS346 event camera.

emulation as a substitute for real cameras in research and development in general. Nevertheless, more extensive and varied tests and analyses should be performed to further validate this proposition. For example, testing on more scenarios and even more tasks may yield more insights, especially since the relatively large number of parameters specifying the camera's behaviour have not been fully explored here. The small decrease in obstacle avoidance performance may in fact be due to a sub-optimal tuning of the DAVIS parameters, in contrast to the emulator's parameters, which underwent a more rigorous refinement process during our tuning phase (section 6.1.2). With regards to our emulation strategy, the processing of the DAVIS's color frames also revealed a degradation in performance due to lower frame rates and lower quality colors, which motivate future analyses of its limitations and subsequent improvements.

7

Conclusions

In this thesis, we proposed, designed, implemented, and extensively evaluated a neuromorphic approach to obstacle avoidance on a robot manipulator with a single on-board camera. Our pipeline transforms visual inputs into corrective obstacle avoidance maneuvers, combining high-level trajectory planning and low-level reactive adjustments using dynamic motion primitives. We utilize event-based vision and spiking neural networks, which provide the neuromorphic sensing and processing in our approach. Simulated and real robot experiments provided evidence that our approach is largely successful in achieving the targeted real-time, online obstacle avoidance behaviour across a variety of task scenarios, clearly proving its utility over a baseline, non-adaptive trajectory planning approach. Additionally, we have gained various notable insights through our experimentation and analyses in this work, which we discuss in this closing chapter.

Firstly, we revisit the questions posed in the introduction of this report and address them in the following.

Are there benefits to using event camera data for obstacle avoidance on a robot? Event-based vision provides the relevant task data, i.e. due to motion, for our SNN to process, which in turn facilitates the derivation of the stable obstacle avoidance maneuvers observed in our experiments. Event data is therefore a key aspect of our approach to obstacle avoidance. This strategy discards irrelevant information about the scene, including colors and stationary objects in the background, thus intuitively reducing wasted bandwidth. Although we mainly operated with emulated event data, we argue that these findings similarly apply to real event data. This was supported by results from preliminary tests with a real event camera, in which similarly successful obstacle avoidance was achieved. Nevertheless, this question could be addressed more fully through comparative evaluations of the event camera with a normal camera, in order to gain insights on hardware-specific advantages of event-based vision, such as reduced power consumption and latencies.

Are there benefits to using SNN processing for obstacle avoidance on a robot? The SNN is the principal component in our pipeline. Its importance is underscored by the results presented in section 6.5, which verify that our approach fails if the SNN is excluded and avoidance behaviour is derived directly from raw events instead. Through the computation time analysis of section 6.1.4, we have shown the dependence of SNN computations on the amount of perceived motion. This result highlights the SNN's

property of performing computations only at relevant inputs and thus the potential savings in energy and time compared to conventional algorithms such as DNNs, in addition to confirming the aforementioned importance of event-based data. Furthermore, the analysis presented in section 6.4 has confirmed the SNN's robustness to differences in event data produced by different methods that is evident from its relatively consistent responses. These responses were fairly similar even for methods containing noisier event data, suggesting that the SNN performs noise filtering that obviates the need for additional noise removal (such as by blurring source images when emulating events). Although these findings confirm the utility of the SNN, a dedicated comparison to a DNN may provide further validation and insights.

Is it feasible to decode SNN outputs into obstacle avoidance behaviour on a robot? Our first challenge was in determining the feasibility of this approach, prior to evaluating its success. From initial results in simulation tests, we were able to confirm that our method of decoding output spiking neuron activations into avoidance accelerations produced the intended trajectory adjustments. The key aspects in this regard were the potential fields approach to deriving acceleration values, in addition to the DMP formulation which facilitated online trajectory adaptations by incorporating these values.

Is it possible to achieve reliable, online obstacle avoidance with the proposed neuromorphic approach? The results presented in simulation experiments indicated that the arm could successfully avoid obstacles in different scenarios and this conclusion was reaffirmed in real robot experiments. Our approach enabled online corrective actions such that the robot could adapt its trajectories and still reach its goal, exhibiting high reliability through consistent task success in all but a few challenging scenarios of imminent collision cases. From a quantitative assessment, the resultant obstacle-aware trajectories minimally deviated from baseline, non-adaptive trajectories while succeeding in obstacle avoidance. Qualitatively, the trajectories were also often predictable, safe, and reasonably smooth to not appear unnatural, despite no explicit optimization of these criteria. Results with different SNN weight values (presented in 6.6) have shown the potential of further optimizations through learning. Therefore, our findings ultimately confirm the success of the proposed approach in achieving obstacle avoidance, and encourage future improvements.

From our experiments, we have observed an adequate handling of unseen tasks and task conditions, such as a particularly dimly lit environment, both in simulation and real scenarios. However, we have also noted that low contrasts between an obstacle and the background could lead to degradations in performance, such as with the well-concealed wooden block compared to the metal bar object in the real experiments. This is not surprising, since we expect visual salience to similarly affect humans' or other animals' ability to perceive an object.

Through our transitioning from simulated to real robot experiments, we have found that our approach transfers well, only requiring the adjustment of a few parameters. These parameters: the motion controller gains and event emission threshold, are expected to require modifications since they depend on the fidelity of the Gazebo simulation's actuation dynamics and visual output, which are sure to deviate from the real world to some degree.

From the preceding discussion on SNNs, we can conclude that the structural properties of SNNs have been instrumental in our approach, even with no synaptic weight adjustments. This matches findings from other works which we have discussed in our literature review concerning the algorithmic utility of SNNs. An interesting interpretation of our results is that the obstacle avoidance performance we observe is due to random, fixed "features" extracted by the SNN from the event data due to the random weight values we sample. These features are able to produce sufficiently robust behaviour.

A principled evaluation methodology and highly interpretable parameters were key to the gradual improvement and final assessment of our implementation. The tuning → validation → testing procedure and the coverage of our task scenario distribution were especially useful for obtaining statistically significant results with which we could improve the performance of our approach. As initially postulated, the utility of simulations for the initial tuning and testing was clear, since the large number of trials would have been impractical to run on the robot (not counting the pre-tuning phase, a total of 5090 simulated trials were executed during the tuning → validation → testing process). The advantages of the simulation were fully realized by automating trial execution, data collection, metric computations, and batch comparisons. The interpretability of our parameters was demonstrated by the successful re-tuning of select parameters to achieve desired characteristics and thus address observed failures in particular scenarios (such as the instantiation of sets 10-12 to improve performance in scenario 2).

The adaptive trajectory representation afforded by the DMPs is evidently an integral part of our pipeline. By using a DMP as an adaptive planner, we address a limitation of purely reactive obstacle avoidance approaches (some of which we have reviewed), which do not actively compensate for avoidance maneuvers in order to return an agent to its original, intended path. Our approach ensures that the arm maintains progress towards its goal throughout its execution, regardless of avoidance maneuvers.

Our event camera emulation strategy has shown that we can successfully mimic event-based vision for the purposes of conducting preliminary evaluations of a pipeline incorporating neuromorphic sensing. Emulation is thus a viable and useful research tool in the absence of a real camera, as evidenced by our preliminary comparisons to the DAVIS346.

The implementation of our pipeline was designed with modularity and generality as core principles. As a result, it can easily be run on a different robot platform, essentially by substituting the Kinova's low-level controller component and adjusting associated parameters. The design therefore promotes future applications of this neuromorphic approach to other problems.

Although we have not used dedicated neuro-processors to run SNNs, this project has provided initial insights on the feasibility and utility of an event-based SNN approach to obstacle avoidance. The results motivate further research on neuromorphic approaches in intelligent robot design.

We draw attention to the limitations of our approach and opportunities for future work in sections 7.1 and 7.2, respectively.

7.1 Limitations

Despite the high success rate and reliability we achieve in obstacle avoidance across task scenarios, the performance of our approach was observed to degrade in two task conditions that bear mentioning.

Firstly, we noted a limited capability of avoiding high-speed obstacles collisions, which could be addressed by increasing sensitivity but at the cost of compromising safety. In particular, a more acute SNN response could be attained by reacting more strongly to the fewer events induced by a fast obstacle such that the arm is driven faster to avoid it. However, the resultant increases in arm velocities and accelerations, if even physically feasible, introduce undesirable safety risks. Therefore, we presently acknowledge a fundamental limitation in reliably reacting to fast obstacles due to physical constraints. Specific bounds on admissible obstacle speeds can be determined through further dedicated studies.

The second task condition that presented a challenge was the dimmer and more cluttered background of the simulated "Office" environment, in which success rate was noticeably affected. The lower illumination levels, though handled well in other backgrounds, are thought to lead to less contrasts between objects (for e.g., the obstacle and the background), which in turn lead to less events in response to motions and consequently lower avoidance velocities. Clutter, on the other hand, may contribute to relatively more background events that saturate the SNN response, possibly overwhelming the localized responses at obstacle positions. Both factors seem to contribute to the failures observed in this environment. However, a more complete understanding of these effects requires further investigation.

We have observed a noteworthy degree of variance in avoidance trajectories in one of the tasks (3), in which some failures were caused by an unfortunate motion direction at the beginning of the trajectory from which subsequent corrections were unlikely to prevent collisions. This motivates further analyses to quantify and study the sensitivity to variations in input stimuli as well as the uncertainties associated with the trajectories produced by our approach. At present, the resultant avoidance maneuvers are fairly consistent but nevertheless exhibit occasional variations in some cases.

Although we have presented a set of well-formulated qualitative evaluation criteria, we have selected specific quantitative measures to evaluate each, such as the estimated angular velocity for predictability. Since the concept of predictability, for example, can be interpreted in myriad ways, a better approach could involve an aggregate of measures for evaluating each of these criteria. In addition, our analysis lacks dedicated user studies for gathering and aggregating subjective opinions on each of these criteria, which may be useful for drawing more general conclusions.

Our real robot experiments consisted of fewer task scenarios than the simulation experiments. This is in part due to the relative ease in conducting large batches of automated trials in simulation (in comparison to real trials), but also due to our methodology of developing and tuning initially in simulation. Nevertheless, we emphasize the fewer real robot experiments conducted in this study and the importance of further experimentation, particularly with a larger variety of scenarios and tasks.

Finally, we note a limitation of the procedure followed in the real robot experiments involving dynamic obstacles, which were manually moved. Although multiple precautions were taken to maximize the repeatability of task conditions (such as measuring the initial and final positions at every trial), a margin of error is expected. As mentioned, any such variations could in fact be beneficial in testing for robustness in a real-life setting, in which precisely, repeated conditions are unlikely. In addition, from the perspective of a quantitative evaluation, the large numbers of trials conducted in these scenarios is likely sufficient for eliminating variances in the performance metrics. For more controlled settings, we could construct

mechanisms for automating obstacle motions, which could potentially minimize positional errors.

7.2 Future work

We have mentioned various possible avenues of future work and extensions throughout this report, which we summarize and elaborate on in this section.

Given the success of our approach with emulated event data and the preliminary results with the DAVIS346, a next step is to further explore and utilize the capabilities of a real event camera. Dedicated experiments could be run to analyse the camera's properties in relation to a conventional camera as a component of our pipeline. These can include robustness to lighting variations, latencies, responses to motion blur, etc.

Other potential experiments concerning event data can apply to both real cameras and emulators. An example is to study the dependence between the distance of objects from the camera and the event activity they induce. A more complete understanding of this relationship could useful in developing or improving methods for filtering irrelevant event data, especially from far away background objects. Another example concerns testing more approaches to event noise filtering; in this work, we have exclusively tested binary erosion filtering. Although we have identified the SNN's noise filtering properties, these event filtering methods could be useful tools for further refining the SNN input data in some cases.

With regards to our neural network, there are various hyperparameters, architectural alternatives, and data representation methods to explore. Chief among these are different architectures and spiking neuron models, which may yield interesting results in our SNN processing pipeline. In addition, we can explore different methods of decoding the output of the SNN; though we have used the first-spike-time temporal code, various other rate, temporal, and population codes (discussed in our background section) can be applied instead. In order to further validate the benefits of the SNN in our pipeline, we could also build and test an ANN-based implementation of our obstacle avoidance pipeline and examine the differences in results.

The potential fields method that we use to decode SNN output neural activations can be implemented in a variety ways which may be worth testing in future experiments. For example, some formulations consider dynamic obstacle velocities, in addition to positions, such as in Park et al. (2008), while others present modifications to the potential field equations (Hua et al. (2019), Mronga et al. (2020)).

A particularly promising extension of this work is to incorporate learning in two areas: adjusting the SNN weights and optimizing the hyperparameters of the pipeline. Given the results of our random weights analysis, we expect that synaptic weight tuning can have a positive impact on obstacle avoidance behaviour. A significant challenge would be the selection of an appropriate learning algorithm. As discussed in our literature review, the best method to train an SNN remains very much an open question, but we could consider STDP and its variants or surrogate gradient methods, for example. Other interesting approaches to SNN learning include liquid-state machines (LSMs) (Ponghiran et al. (2019)) and the general concept of applying a simple rule such as linear regression on a representation of output spiking activity (Michaelis et al. (2020)). Our manual hyperparameter tuning procedure could be substituted by a learning or optimization algorithm, such as reinforcement learning or evolutionary optimization. Such an

approach avoids the tedium of manual tuning and could potentially yield more optimal parameter values.

As mentioned in the previous section, the qualitative evaluation we have performed can be improved in future extensions. Primarily for the more subjective criteria, such as naturalness, we can conduct user studies in which executions are evaluated by naïve subjects through questionnaires designed to elucidate the general perception of the robot's behaviour.

We also noted the uncertainty in trajectory outcomes of our approach, observed in one of the simulation tasks, as a possible target for future investigations. The slight variations in results that lead to this uncertainty are due to the complex interactions and data flow within the several stages of our pipeline. In order to better understand these interactions, we can formulate experiments designed to quantify the effects of input variations, the results of which may explain the observed differences in trajectories.

As explained in the introduction, we have deliberately only considered collisions with the robot's end-effector throughout this work. Future extensions could include somehow augmenting our trajectory adaptation approach to incorporate avoidance of collisions with the rest of the robot's body.

In our implementation, the DMP is currently a part of the motion control component, and it can be separated and implemented in a dedicated component in future modifications. One of the advantages to this modification is the possibility of incorporating and testing different approaches to planning adapted trajectories instead of the DMP, if any.

As previously discussed, the full extent to which the SNN in our approach provides processing speed, power consumption, and other improvements can be studied only when the networks are run on neuromorphic hardware. Naturally, a potential extension of our work is thus to explore the integration of a neuromorphic processor as a next step towards a more fully neuromorphic processing approach.

Finally, we have demonstrated our approach here in the domain of robot manipulation. In future work, we can apply our implementation to a navigation scenario, for example, in which the avoidance of obstacles is also a relevant problem. In that case, the navigation planner may significantly differ from the DMP and thus necessitate adapting the relevant aspects of the pipeline. The results of experiments in a different application domain can provide further validation of the neuromorphic concept we have developed and evaluated in this thesis.

A

Comparison of Consumption-to-Computation Ratios

The following table lists two supercomputers whose processing power (in FLOPS) is reportedly comparable to that of the human brain, the processor in the laptop used to write this, and the current most powerful GPU. Given estimations of consumption (in W) and computing power (in petaFLOPS), the ratio of the two is calculated to show the human brain's relative efficiency (the smallest ratio).

In the following, we compare the ratio of power consumption to computational power of the human brain with two supercomputers whose processing power is reportedly comparable to that of the human brain, the processor powering the laptop used to write this report, and the current most powerful GPU (at the time of writing). This ratio is a measure of efficiency and is used to show the human brain's superior efficiency (since it has the smallest ratio). The computational power is expressed in terms of FLOPS[1].

We compute the ratio by dividing the power rating **(in W)** by the computational power **(in petaFLOPS)**[2].

The results are shown in Table A.1. Table A.2 lists the sources consulted for the statistics of all devices.

Table A.1: A comparison of the human brain and various computers based on estimates of computational power and power consumption. (References in Table A.2.)

Device	**Computing power (FLOPS)**	**Power Consumption (W)**	**Consumption-to-Computation Ratio (W/petaFLOPS)**
Human Brain	$[20.0 - 38.0] \times 10^{15}$	20.0	0.5-1.0
IBM's Sequoia	16.3×10^{15}	7.9×10^{6}	484068.6
Tianhe-2 (TH-2)	33.0×10^{15}	17.9×10^{6}	539393.94
AMD Ryzen 7 4800H	244.5×10^{9}	45.0	184049.1
NVIDIA Titan V	0.1×10^{15}	600.0	6000.0

[1] Floating-point operations per second

[2] Unit magnitudes are chosen for representational convenience.

Table A.2: Sources for the statistics presented in Table A.1

Device	**Source**
Human brain	- Dharmendra Modha, Director of DARPA's SyNAPSE Project: https://blogs.scientificamerican.com/news-blog/computers-have-a-lot-to-learn -from-2009-03-10/ - Ray Kurzweil's book: Kurzweil, R. (2000). The age of spiritual machines: When computers exceed human intelligence. Penguin.
IBM's Sequoia	https://arstechnica.com/information-technology/2012/06/with-16-petaflops-and-1-6m-cores-doe-supercomputer-is-worlds-fastest/
Tianhe-2 (TH-2)	https://top500.org/news/china-tops-supercomputer-rankings-with-new-93-petaflop-machine/
AMD Ryzen 7 4800H	- https://www.amd.com/en/products/apu/amd-ryzen-7-4800h - https://gadgetversus.com/processor/amd-ryzen-7-4800h-gflops-performance/
NVIDIA Titan V GPU	https://www.nvidia.com/en-us/titan/titan-v/

B

Samples of Configuration and Metrics Files

```yaml
motion_controller:
  model_state_read_frequency: 10
  goal_reaching_tol: 0.03
  position_reaching_tol: 0.02
  per_dim_reaching_tol: 0.005
  goal_reaching_timeout: 60.0
  position_reaching_timeout: 10.0
  obs_avoidance_dist_tol: 0.1
  pf_method: park
    method_params:
      p_0: 20
      eta: 300
      grad_factor: 1500
      max_phi: 4000
  fst_activation_threshold_factor: 14
  K_p: 5.0
  K_d: 10.0
  K_i: 0.0
  aggregate_phi: true
  obstacle_dims:
    x: 0.1
    y: 0.1
    z: 0.1
## ...
event_image_streamer:
  theta: 30.0
  compute_from_rgb: true
  record_off_events: true
  register_off_events_as_on: true
  source_type: ros_topic
  image_topic: /camera/color/image_raw
```

```yaml
snn_simulator:
  sim_time: 20
  snn_thresh: -52.0
  snn_reset: -62.0
  snn_rest: -62.0
  snn_refrac: 5
  snn_decay: 20
  snn_thresh_increase_at_spike: 0.05
  snn_thresh_decay: 10000000
  init_weight_factor: 7.0
  input_shape:
    - 1
    - 120
    - 160
  layers:
    layer_1:
      type: "DiehlAndCookNodesModded"
      name: "H"
      kernel_size:
        - 8
        - 8
      stride_size:
        - 4
        - 4
      padding_size:
        - 0
        - 0
      output_tensor_shape:
        - 1
        - 29
        - 39
    layer_2:
      type: "DiehlAndCookNodesModded"
      name: "Y"
      kernel_size:
        - 4
        - 4
      stride_size:
        - 2
        - 2
      padding_size:
        - 0
        - 0
      output_tensor_shape:
        - 1
        - 13
        - 18
```

Listing 3: Samples of configuration files for the `motion_controller`, `event_image_streamer`, and `snn_simulator`

```yaml
snn_simulator:
  ## ...
  input_shape:
    - 1
    - 120
    - 160
  layers:
    layer_1:
      type: "DiehlAndCookNodesModded"
      name: "Y"
      kernel_size:
        - 8
        - 8
      stride_size:
        - 4
        - 4
      padding_size:
        - 2
        - 2
      output_tensor_shape:
        - 1
        - 30
        - 40
```

Listing 4: A portion of the `snn_simulator` parameters in set 13, which depicts the one-layer SNN specification.

```yaml
- model_name: box
  link_name: box_link
  start:
    - 1.05
    - -0.35
    - 1.05
  end:
    - 0.4
    - 0.25
    - 1.05
  delta: 0.002
```

Listing 5: A sample of a `gazebo_object_animator` configuration file. This file defines the trajectory followed by a dynamic obstacle (in this case, a box) in a given task scenario.

```yaml
execution_time: 11.72734
trajectory_length: 0.93867
nominal_trajectory_length: 0.52835
collisions: 0
dist_to_goal: 0.02379
success: 1
```

```yaml
Params. 1:
  num_trials: 50
  statistics:
    execution_time:
      num_data_points: 50
      Mean: 16.598157439231873
      Min: 8.006410121917725
      Q1: 12.371076822280884
      Median: 16.69411051273346
      Q3: 19.563407063484192
      Max: 28.73162293434143
      Std: 5.2464288076647065
      Outliers: []
    collisions:
      num_data_points: 50
      Mean: 0.38
      Min: 0
      Q1: 0.0
      Median: 0.0
      Q3: 0.0
      Max: 12
      Std: 1.8170395298565254
      Outliers:
      - 4
      - 12
      - 3
    success:
      num_data_points: 50
      Mean: 0.4
      Min: 0
      Q1: 0.0
      Median: 0.0
      Q3: 1.0
      Max: 1
      Std: 0.49487165930539356
      Outliers: []
Params. 2:
# ...
```

Listing 6: Samples of YAML files produced by our automated evaluation pipeline for: i) a single trial metrics (left) and ii) aggregate metrics for batches of trials from associated with different parameter sets (right). (Note that we omitted some quantitative metrics from the batch statistics example and only included one parameter set for brevity.)

C

Parameter Sets

The tables in this appendix contain the parameter values of every parameter set described in section 6.1, which are organized as follows. The first two tables provide the values chosen for all parameters in the pre-tuning phase; table C.1 contains the parameters that are then fixed, while table C.2 contains the subset of the parameters that are tuned in the next phase.

Tables Tables C.3–C.16 contain sets 1-13. For brevity, each table lists the parameters that have been changed up until that point in the tuning phase. Parameter values that were changed in a given set compared to the last are in boldface.

Table C.1: The subset of parameters that have been fixed before the tuning, validation, and testing procedure. These values have been chosen at the initial pre-tuning phase.

Component	Parameter	Value
event_image_streamer	compute_from_rgb	True
	rgb_multi_channel	True
	record_off_events	True
	register_off_events_as_on	True
	source_type	ros_topic
	image_topic	/camera/color/image_raw
snn_simulator	snn_reset	-62.0
	snn_rest	-62.0
	snn_thresh_increase_at_spike	0.05
	snn_thresh_decay	10000000
motion_controller	model_state_read_frequency	10
	goal_reaching_tol	0.03
	goal_reaching_timeout	60.0
	position_reaching_timeout	10.0
	pf_method	park
	p_0	30
	aggregate_phi	true
	K_p	5.0
	K_i	10.0
	K_d	0.0

Table C.2: The subset of parameters that were tuned during the tuning phase. These values have been chosen after the initial pre-tuning phase.

Component	Parameter	Value
event_image_streamer	theta	30
	filtering_strategy	None
snn_simulator	init_weight_factor	7.0
	sim_time	20
	snn_thresh	-52.0
	snn_refrac	5
	snn_decay	20
motion_controller	position_reaching_tol	0.02
	obs_avoidance_dist_tol	0.1
	max_phi (Park)	4000
	eta (Park)	300
	grad_factor (Park)	1500
	phi_horizon	Inf
	safety_strategy	None
	position_limits	None
	motion_loop_frequency	33
	fst_activation_threshold_factor	14

Table C.3: Parameter Set 1

Component	Parameter	Value
event_image_streamer	theta	30
	filtering_strategy	None
snn_simulator	init_weight_factor	7.0
motion_controller	position_reaching_tol	0.02
	obs_avoidance_dist_tol	0.1
	max_phi (park)	4000
	phi_horizon	Inf
	safety_strategy	1
	safety_tol	0.4
	vel_reduction_factor	0.6
	phi_reduction_factor	0.2
	position_limits	None

Table C.4: Parameter Set 2

Component	Parameter	Value
event_image_streamer	theta	30
	filtering_strategy	None
snn_simulator	init_weight_factor	7.0
motion_controller	position_reaching_tol	0.02
	obs_avoidance_dist_tol	0.1
	max_phi (park)	4000
	phi_horizon	Inf
	safety_strategy	1
	safety_tol	0.4
	vel_reduction_factor	0.6
	phi_reduction_factor	0.6
	position_limits	None

Table C.5: Parameter Set 3

Component	Parameter	Value
event_image_streamer	theta	30
	filtering_strategy	None
snn_simulator	init_weight_factor	7.0
motion_controller	position_reaching_tol	0.02
	obs_avoidance_dist_tol	0.1
	max_phi (park)	4000
	phi_horizon	Inf
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.6
	position_limits	None

Table C.6: Parameter Set 4

Component	Parameter	Value
event_image_streamer	theta	30
	filtering_strategy	None
snn_simulator	init_weight_factor	7.0
motion_controller	position_reaching_tol	0.02
	obs_avoidance_dist_tol	0.1
	max_phi (park)	4000
	phi_horizon	Inf
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	position_limits	None

Table C.7: Parameter Set 5

Component	Parameter	Value
event_image_streamer	theta	30
	filtering_strategy	None
snn_simulator	init_weight_factor	7.0
motion_controller	position_reaching_tol	0.02
	obs_avoidance_dist_tol	0.1
	max_phi (park)	4000
	phi_horizon	Inf
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	position_limits: z	[0.87, Inf]

Table C.8: Parameter Set 6

Component	Parameter	Value
event_image_streamer	theta	30
	filtering_strategy	binary_erosion
	structure_size	(3, 3)
snn_simulator	init_weight_factor	7.0
motion_controller	position_reaching_tol	0.02
	obs_avoidance_dist_tol	0.1
	max_phi (park)	4000
	phi_horizon	Inf
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	position_limits: z	[0.87, Inf]

Table C.9: Parameter Set 7

Component	Parameter	Value
event_image_streamer	theta	30
	filtering_strategy	binary_erosion
	structure_size	(3, 3)
snn_simulator	**init_weight_factor**	15.0
motion_controller	position_reaching_tol	0.02
	obs_avoidance_dist_tol	0.1
	max_phi (park)	4000
	phi_horizon	Inf
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	position_limits: z	[0.87, Inf]

Table C.10: Parameter Set 8

Component	Parameter	Value
event_image_streamer	theta	30
	filtering_strategy	binary_erosion
	structure_size	(5, 5)
snn_simulator	**init_weight_factor**	20.0
motion_controller	position_reaching_tol	0.02
	obs_avoidance_dist_tol	0.1
	max_phi (park)	4000
	phi_horizon	Inf
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	position_limits: z	[0.87, Inf]

Table C.11: Parameter Set 9

Component	Parameter	Value
event_image_streamer	theta	30
	filtering_strategy	binary_erosion
	structure_size	(4, 4)
snn_simulator	init_weight_factor	20.0
motion_controller	**position_reaching_tol**	0.01
	obs_avoidance_dist_tol	0.1
	max_phi (park)	5000
	phi_horizon	2
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	safety_strategy	2
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.8
	position_limits: z	[0.87, Inf]

Table C.12: Parameter Set 10

Component	Parameter	Value
event_image_streamer	theta	28
	filtering_strategy	binary_erosion
	structure_size	(5, 5)
snn_simulator	init_weight_factor	20.0
motion_controller	position_reaching_tol	0.01
	obs_avoidance_dist_tol	0.15
	max_phi (park)	4000
	phi_horizon	12
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	safety_strategy	2
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	position_limits: z	[0.87, Inf]

Table C.13: Parameter Set 11

Component	Parameter	Value
event_image_streamer	theta	28
	filtering_strategy	binary_erosion
	structure_size	(5, 5)
snn_simulator	**init_weight_factor**	25.0
motion_controller	position_reaching_tol	0.01
	obs_avoidance_dist_tol	0.15
	max_phi (park)	4000
	phi_horizon	12
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	safety_strategy	2
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	position_limits: z	[0.87, Inf]

Table C.14: Parameter Set 12

Component	Parameter	Value
event_image_streamer	theta	28
	filtering_strategy	binary_erosion
	structure_size	(5, 5)
snn_simulator	**init_weight_factor**	20.0
motion_controller	position_reaching_tol	0.01
	obs_avoidance_dist_tol	0.15
	max_phi (park)	4000
	phi_horizon	12
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	safety_strategy	2
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.6
	position_limits: z	[0.87, Inf]

Table C.15: Parameter Set 13

Component	Parameter	Value
event_image_streamer	theta	28
	filtering_strategy	binary_erosion
	structure_size	(5, 5)
snn_simulator	**init_weight_factor**	2000.0
	sim_time	3
	snn_thresh	-58.0
	snn_refrac	0
	snn_decay	1000
motion_controller	position_reaching_tol	0.01
	obs_avoidance_dist_tol	0.15
	max_phi (park)	8000
	eta (Park)	600
	grad_factor (Park)	40000
	phi_horizon	12
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	safety_strategy	2
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.6
	position_limits: z	[0.87, Inf]

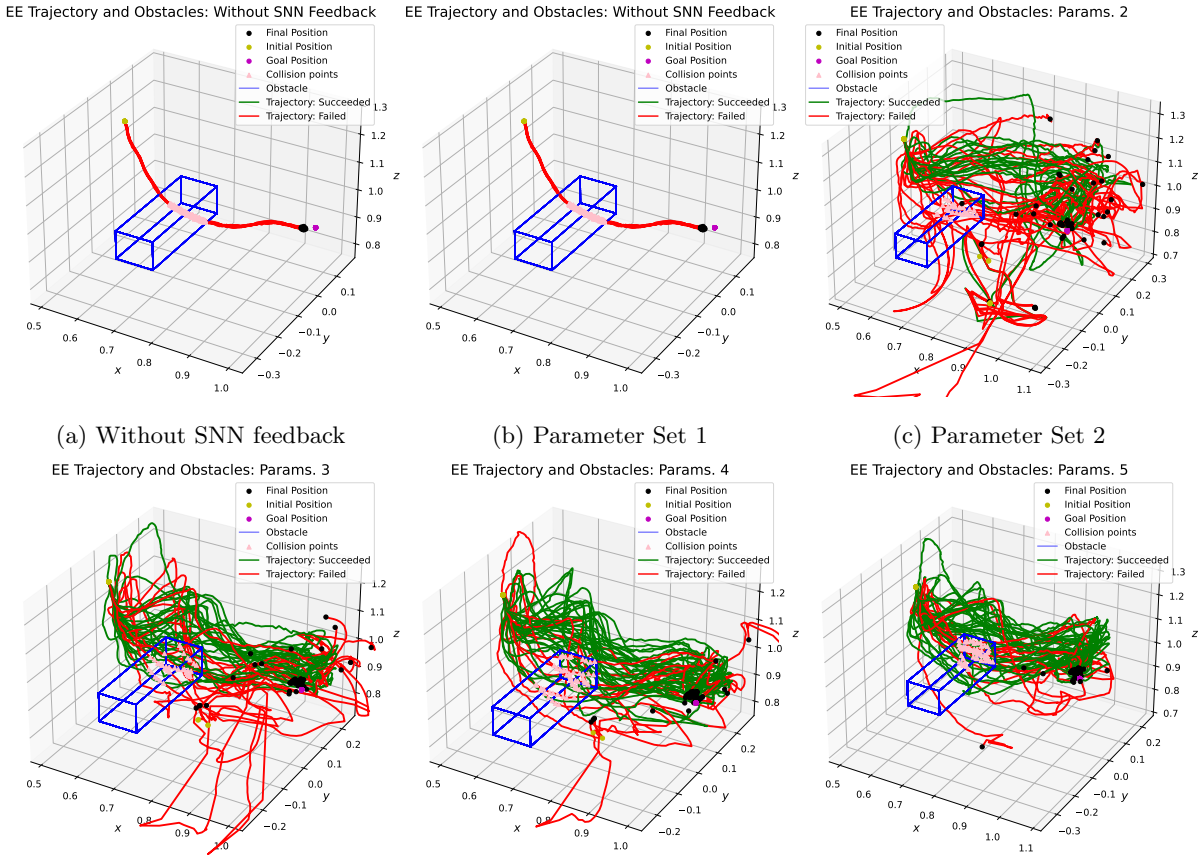
Table C.16: Parameter Set 14

Component	Parameter	Value
event_image_streamer	theta	28
	filtering_strategy	binary_erosion
	structure_size	(5, 5)
snn_simulator	**init_weight_factor**	2000.0
	sim_time	3
	snn_thresh	-58.0
	snn_refrac	0
	snn_decay	1000
motion_controller	position_reaching_tol	0.01
	obs_avoidance_dist_tol	0.15
	max_phi (park)	6000
	eta (Park)	600
	grad_factor (Park)	10000
	phi_horizon	12
	safety_strategy	1
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.4
	safety_strategy	2
	safety_tol	0.2
	vel_reduction_factor	0.8
	phi_reduction_factor	0.6
	position_limits: z	[0.87, Inf]

D

Extra Visualizations

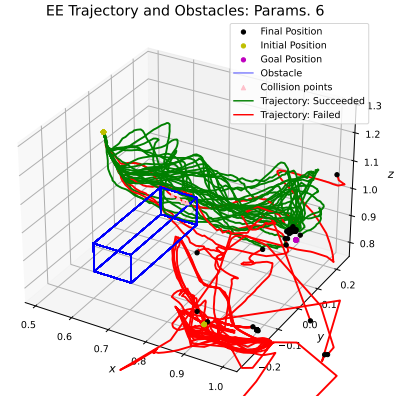
D.1 Tuning Phase: Trajectories Executed in Scenario 3 (Sets 1-12)



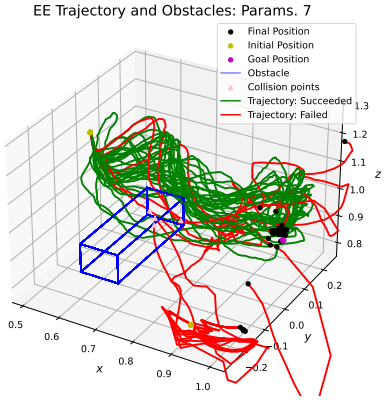
(a) Without SNN feedback (b) Parameter Set 1 (c) Parameter Set 2

(d) Parameter Set 3 (e) Parameter Set 4 (f) Parameter Set 5

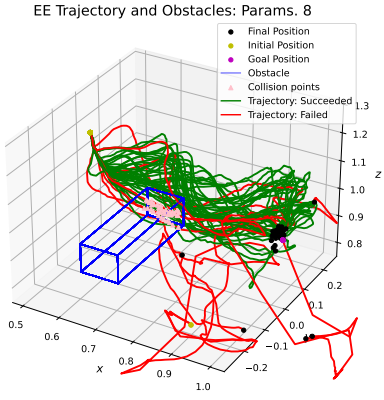
Figure D.1: Trajectories executed in tuning scenario 3: without SNN feedback and parameter sets 1-5.



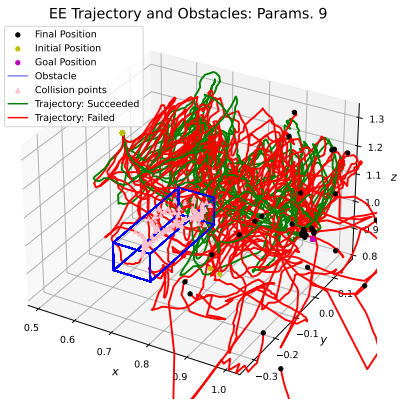
(a) Parameter Set 6



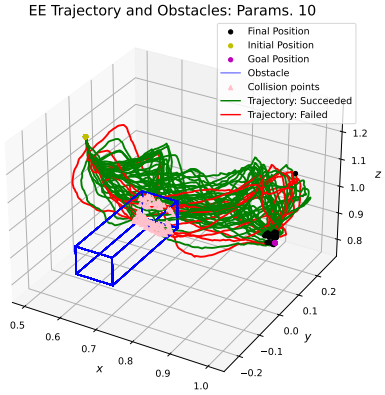
(b) Parameter Set 7



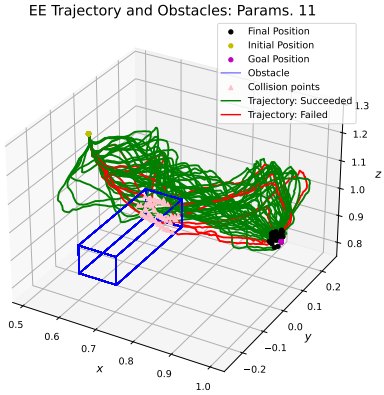
(c) Parameter Set 8



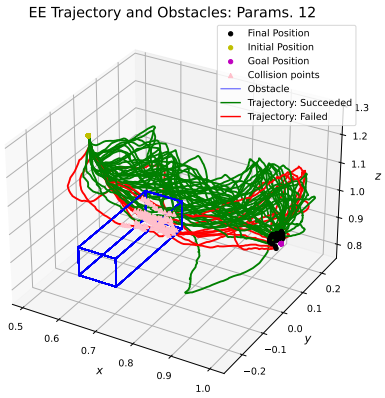
(d) Parameter Set 9



(e) Parameter Set 10



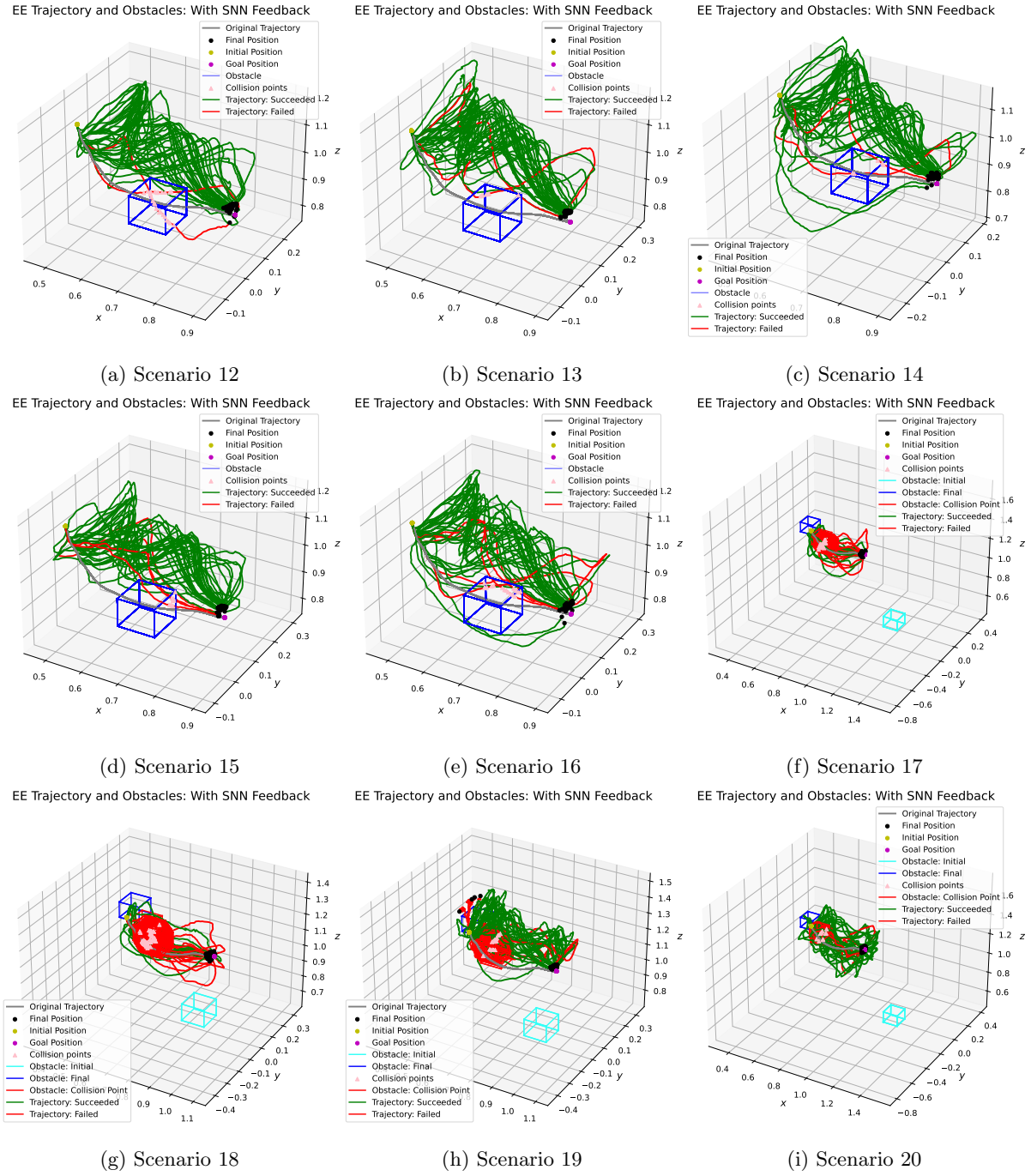
(f) Parameter Set 11



(g) Parameter Set 12

Figure D.2: Trajectories executed in tuning scenario 3: parameter sets 6-12.

D.2 Testing Phase: Trajectories Executed in Scenarios 12-29

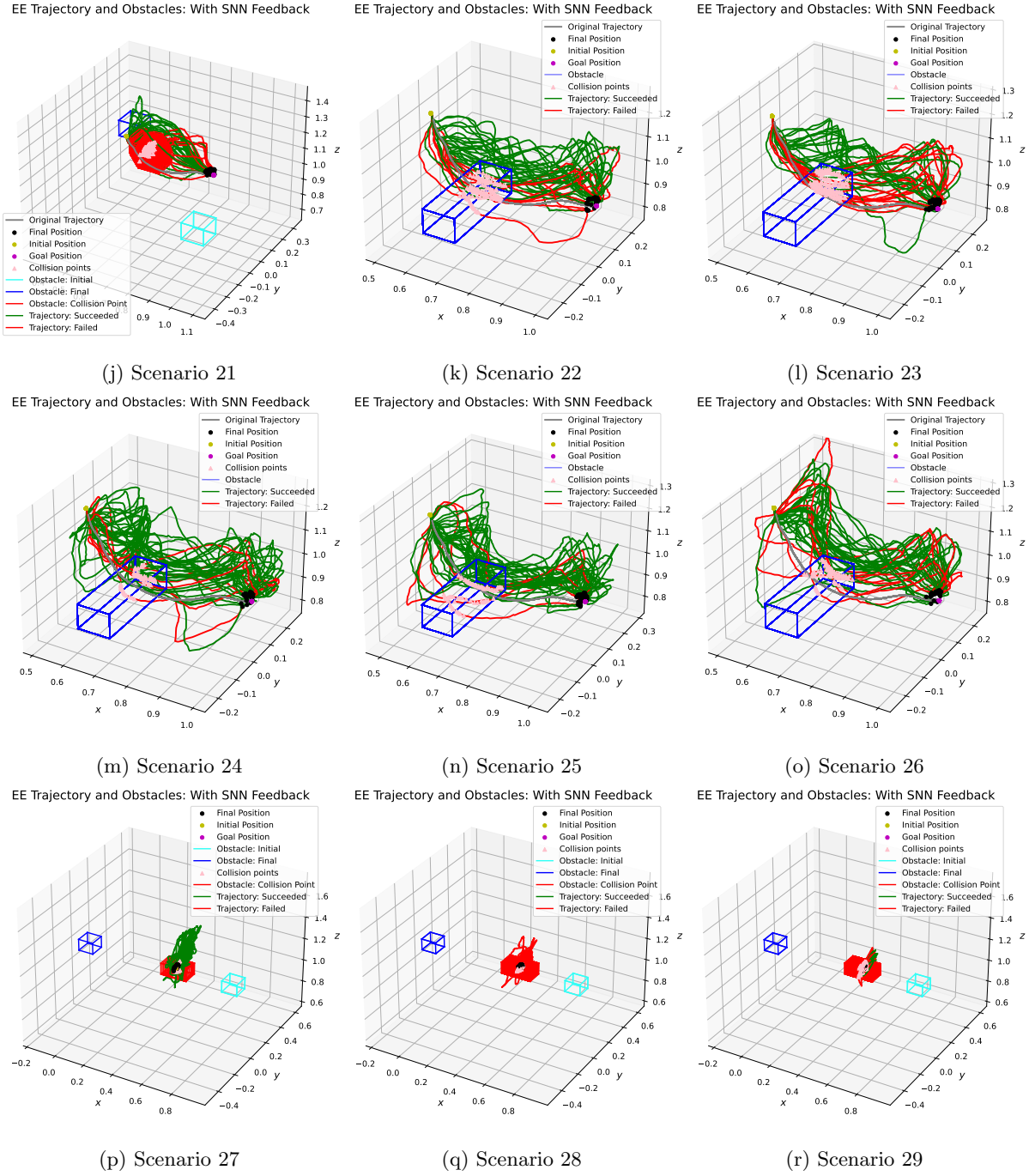


(a) Scenario 12 (b) Scenario 13 (c) Scenario 14

(d) Scenario 15 (e) Scenario 16 (f) Scenario 17

(g) Scenario 18 (h) Scenario 19 (i) Scenario 20

Figure D.3: Trajectories executed in testing scenarios, with SNN feedback: testing scenarios 12-20.



(j) Scenario 21 (k) Scenario 22 (l) Scenario 23

(m) Scenario 24 (n) Scenario 25 (o) Scenario 26

(p) Scenario 27 (q) Scenario 28 (r) Scenario 29

Figure D.3: Trajectories executed in testing scenarios, with SNN feedback (cont.): testing scenarios 21-29.

References

Abdelrahman, A. F., Mitrevski, A., & Plöger, P. G. (2020). Context-aware task execution using apprenticeship learning. In *2020 ieee international conference on robotics and automation (icra)* (pp. 1329–1335).

Aguilar, W. G., Casaliglla, V. P., & Pólit, J. L. (2017). Obstacle avoidance based-visual navigation for micro aerial vehicles. *Electronics*, *6*(1), 10.

Ananthaswamy, A. (2021). *Artificial Neural Nets Finally Yield Clues to How Brains Learn.* Retrieved from `https://www.quantamagazine.org/artificial-neural-nets-finally-yield-clues-to-how-brains-learn-20210218/`

Arakawa, R., & Shiba, S. (2020). Exploration of reinforcement learning for event camera using car-like robots. *arXiv preprint arXiv:2004.00801*.

Bartolozzi, C., Indiveri, G., & Donati, E. (2022). Embodied neuromorphic intelligence. *Nature communications*, *13*(1), 1–14.

Becanovic, V., Bredenfeld, A., & Ploger, P. G. (2002). Reactive robot control using optical analog vlsi sensors. In *Proceedings 2002 ieee international conference on robotics and automation (cat. no. 02ch37292)* (Vol. 2, pp. 1223–1228).

Bečanović, V., Indiveri, G., Kobialka, H.-U., Plöger, P. G., & Stocker, A. (2002). Silicon retina sensing guided by omni-directional vision. In *Proc. ninth ieee conf. on mechatronics and machine vision in practice (m2vip)* (pp. 10–12).

Beckert, D., Pereira, A., & Althoff, M. (2017). Online verification of multiple safety criteria for a robot trajectory. In *2017 ieee 56th annual conference on decision and control (cdc)* (pp. 6454–6461).

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., ... Eliasmith, C. (2014). Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, *7*, 48.

Beyeler, M., Carlson, K. D., Chou, T.-S., Dutt, N., & Krichmar, J. L. (2015). Carlsim 3: A user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks. In *2015 international joint conference on neural networks (ijcnn)* (pp. 1–8).

Bing, Z., Meschede, C., Huang, K., Chen, G., Rohrbein, F., Akl, M., & Knoll, A. (2018). End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle. In *2018 ieee international conference on robotics and automation (icra)* (pp. 4725–4732).

Bing, Z., Meschede, C., Röhrbein, F., Huang, K., & Knoll, A. C. (2018). A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in neurorobotics*, *12*, 35.

Blouw, P., & Eliasmith, C. (2020). Event-driven signal processing with neuromorphic computing systems. In *Icassp 2020-2020 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 8534–8538).

Borenstein, J., Koren, Y., et al. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, *7*(3), 278–288.

Bouvier, M., Valentian, A., Mesquida, T., Rummens, F., Reyboz, M., Vianello, E., & Beigne, E. (2019). Spiking neural networks hardware implementations and challenges: A survey. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *15*(2), 1–35.

Brandli, C., Berner, R., Yang, M., Liu, S.-C., & Delbruck, T. (2014). A 240x180 130 db 3us latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, *49*(10), 2333–2341.

Brock, O., & Khatib, O. (2002). Elastic strips: A framework for motion generation in human environments. *The International Journal of Robotics Research*, *21*(12), 1031–1052.

Ceolini, E., Frenkel, C., Shrestha, S. B., Taverni, G., Khacef, L., Payvand, M., & Donati, E. (2020). Hand-gesture recognition based on emg and event-based camera sensor fusion: A benchmark in neuromorphic computing. *Frontiers in neuroscience*, *14*, 637.

Chen, G., Cao, H., Conradt, J., Tang, H., Rohrbein, F., & Knoll, A. (2020). Event-based neuromorphic vision for autonomous driving: a paradigm shift for bio-inspired visual sensing and perception. *IEEE Signal Processing Magazine*, *37*(4), 34–49.

Chen, H., Liu, W., Goel, R., Lua, R. C., Mittal, S., Huang, Y., ... Patel, A. B. (2019). Fast retinomorphic event-driven representations for video gameplay and action recognition. *IEEE Transactions on Computational Imaging*, *6*, 276–290.

Chen, K., Hwu, T., Kashyap, H. J., Krichmar, J. L., Stewart, K., Xing, J., & Zou, X. (2020). Neurorobots as a means toward neuroethology and explainable ai. *Frontiers in Neurorobotics*, 70.

Chiriatti, G., Palmieri, G., Scoccia, C., Palpacelli, M. C., & Callegari, M. (2021). Adaptive obstacle avoidance for a class of collaborative robots. *Machines*, *9*(6), 113.

Cowley, H. P., Natter, M., Gray-Roncal, K., Rhodes, R. E., Johnson, E. C., Drenkow, N., ... Gray-Roncal, W. (2022). A framework for rigorous evaluation of human performance in human and machine learning comparison studies. *Scientific Reports*, *12*(1), 1–11.

Crick, F. (1989). The recent excitement about neural networks. *Nature*, *337*(6203), 129–132.

Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., ... Risbud, S. R. (2021). Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, *109*(5), 911–934.

Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., ... Yger, P. (2009). Pynn: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, *2*, 11.

Diehl, P. U., & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, *9*, 99.

Dietsche, A., Cioffi, G., Hidalgo-Carrió, J., & Scaramuzza, D. (2021). Powerline tracking with event cameras. In *2021 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 6990–6997).

Drubach, D. (2000). *The brain explained*. Pearson.

Dubeau, E., Garon, M., Debaque, B., de Charette, R., & Lalonde, J.-F. (2020). Rgb-de: Event camera calibration for fast 6-dof object tracking. In *2020 ieee international symposium on mixed and augmented reality (ismar)* (pp. 127–135).

Dumesnil, E., Beaulieu, P.-O., & Boukadoum, M. (2016). Robotic implementation of classical and operant

conditioning as a single stdp learning process. In *2016 international joint conference on neural networks (ijcnn)* (pp. 5241–5247).

Dupeyroux, J., Hagenaars, J. J., Paredes-Vallés, F., & de Croon, G. C. (2021). Neuromorphic control for optic-flow-based landing of mavs using the loihi processor. In *2021 ieee international conference on robotics and automation (icra)* (pp. 96–102).

D'Silva, T., & Miikkulainen, R. (2009). Learning dynamic obstacle avoidance for a robot arm using neuroevolution. *Neural processing letters*, *30*(1), 59–69.

Escobedo, C., Strong, M., West, M., Aramburu, A., & Roncone, A. (2021). Contact anticipation for physical human–robot interaction with robotic manipulators using onboard proximity sensors. In *2021 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 7255–7262).

Fairhall, A. L., Lewen, G. D., Bialek, W., & de Ruyter van Steveninck, R. R. (2001). Efficiency and ambiguity in an adaptive neural code. *Nature*, *412*(6849), 787–792.

Falanga, D., Kleber, K., & Scaramuzza, D. (2020). Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, *5*(40), eaaz9712.

Falotico, E., Vannucci, L., Ambrosano, A., Albanese, U., Ulbrich, S., Vasquez Tieck, J. C., ... others (2017). Connecting artificial brains to robots in a comprehensive simulation framework: the neurorobotics platform. *Frontiers in neurorobotics*, *11*, 2.

Feng, P., Zou, J., Li, H., & Gao, S. (2020). An obstacle avoidance method for autonomous vehicle in straight road based on expanded circle. In *2020 asia-pacific conference on image processing, electronics and computers (ipec)* (pp. 43–46).

Fidjeland, A. K., Roesch, E. B., Shanahan, M. P., & Luk, W. (2009). Nemo: a platform for neural modelling of spiking neurons using gpus. In *2009 20th ieee international conference on application-specific systems, architectures and processors* (pp. 137–144).

Firestone, C. (2020). Performance vs. competence in human–machine comparisons. *Proceedings of the National Academy of Sciences*, *117*(43), 26562–26571.

Flash, T., & Hogan, N. (1985). The coordination of arm movements: an experimentally confirmed mathematical model. *Journal of neuroscience*, *5*(7), 1688–1703.

Fligge, N., McIntyre, J., & van der Smagt, P. (2012). Minimum jerk for human catching movements in 3d. In *2012 4th ieee ras & embs international conference on biomedical robotics and biomechatronics (biorob)* (pp. 581–586).

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, *4*(1), 23–33.

Frémaux, N., & Gerstner, W. (2016). Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in neural circuits*, *9*, 85.

Frenkel, C., Legat, J.-D., & Bol, D. (2020). A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas. In *2020 ieee international symposium on circuits and systems (iscas)* (pp. 1–5).

Furber, S. (2016). Large-scale neuromorphic computing systems. *Journal of neural engineering*, *13*(5), 051001.

Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., ... Scaramuzza, D. (2022, jan). Event-based vision: A survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, *44*(01), 154-180. doi: 10.1109/TPAMI.2020.3008413

Garain, A., Basu, A., Giampaolo, F., Velasquez, J. D., & Sarkar, R. (2021). Detection of covid-19 from ct scan images: A spiking neural network-based approach. *Neural Computing and Applications*, *33*(19), 12591–12604.

García, G. P., Camilleri, P., Liu, Q., & Furber, S. (2016). pydvs: An extensible, real-time dynamic vision sensor emulator using off-the-shelf hardware. In *2016 ieee symposium series on computational intelligence (ssci)* (pp. 1–7).

Gasparetto, A., Boscariol, P., Lanzutti, A., & Vidoni, R. (2015). Path planning and trajectory planning algorithms: A general overview. *Motion and operation planning of robotic systems*, 3–27.

Gerstner, W. (1995). Time structure of the activity in neural network models. *Physical review E*, *51*(1), 738.

Gewaltig, M.-O., & Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia*, *2*(4), 1430.

Göltz, J., Kriener, L., Baumbach, A., Billaudelle, S., Breitwieser, O., Cramer, B., ... others (2021). Fast and energy-efficient neuromorphic deep learning with first-spike times. *Nature machine intelligence*, *3*(9), 823–835.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Goodman, D. F., & Brette, R. (2008). Brian: a simulator for spiking neural networks in python. *Frontiers in neuroinformatics*, *2*, 5.

Hazan, H., Saunders, D. J., Khan, H., Patel, D., Sanghavi, D. T., Siegelmann, H. T., & Kozma, R. (2018). Bindsnet: A machine learning-oriented spiking neural networks library in python. *Frontiers in neuroinformatics*, 89.

Heeger, D., et al. (2000). Poisson model of spike generation. *Handout, University of Standford*, *5*(1-13), 76.

Hodgkin, A. L., & Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, *117*(4), 500.

Hu, Y., Liu, S.-C., & Delbruck, T. (2021). v2e: From video frames to realistic dvs events. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 1312–1321).

Hua, M., Nan, Y., & Lian, S. (2019). Small obstacle avoidance based on rgb-d semantic segmentation. In *Proceedings of the ieee/cvf international conference on computer vision workshops* (pp. 0–0).

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural networks*, *21*(4), 642–653.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., & Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, *25*(2), 328–373.

Illing, B., Gerstner, W., & Brea, J. (2019). Biologically plausible deep learning—but how far can we go with shallow networks? *Neural Networks*, *118*, 90–101.

Indiveri, G. (2021). Introducing 'neuromorphic computing and engineering'. *Neuromorphic Computing and Engineering*, *1*(1), 010401.

Indiveri, G., & Liu, S.-C. (2015). Memory and information processing in neuromorphic systems. *Proceedings of the IEEE*, *103*(8), 1379–1397.

ISO 10218-1:2011. (2011). *Robots and robotic devices - safety requirements for industrial robots - part 1: Robots*. International Organization for Standardization.

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on neural networks*, *14*(6), 1569–1572.

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, *15*(5), 1063–1070.

Jang, H., Simeone, O., Gardner, B., & Gruning, A. (2019). An introduction to probabilistic spiking neural networks: Probabilistic models, learning rules, and applications. *IEEE Signal Processing Magazine*, *36*(6), 64–77.

Joubert, D., Marcireau, A., Ralph, N., Jolley, A., van Schaik, A., & Cohen, G. (2021). Event camera simulator improvements via characterized parameters. *Frontiers in Neuroscience*, 910.

Kasabov, N. K. (2014). Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Networks*, *52*, 62–76.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles* (pp. 396–404). Springer.

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., & Masquelier, T. (2018). Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, *99*, 56–67.

Kheradpisheh, S. R., & Masquelier, T. (2020). Temporal backpropagation for spiking neural networks with one spike per neuron. *International Journal of Neural Systems*, *30*(06), 2050027.

Kim, S., Park, S., Na, B., & Yoon, S. (2020). Spiking-yolo: spiking neural network for energy-efficient object detection. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 34, pp. 11270–11277).

Korteling, J. H., van de Boer-Visschedijk, G., Blankendaal, R. A., Boonekamp, R., & Eikelboom, A. (2021). Human-versus artificial intelligence. *Frontiers in artificial intelligence*, *4*, 622364.

Kwisthout, J., & Donselaar, N. (2020). On the computational power and complexity of spiking neural networks. In *Proceedings of the neuro-inspired computational elements workshop* (pp. 1–7).

Lee, C., Sarwar, S. S., Panda, P., Srinivasan, G., & Roy, K. (2020). Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in neuroscience*, 119.

Lee, H., Ho, H., & Zhou, Y. (2021). Deep learning-based monocular obstacle avoidance for unmanned aerial vehicle navigation in tree plantations. *Journal of Intelligent & Robotic Systems*, *101*(1), 1–18.

LeVine, S. (2017). *Artificial intelligence pioneer says we need to start over.* Retrieved from `https://www.axios.com/ai-pioneer-advocates-starting-over-2485537027.html`

Lichtsteiner, P., Posch, C., & Delbruck, T. (2008). A 128x128 120 db 15us latency asynchronous temporal contrast vision sensor. *IEEE journal of solid-state circuits*, *43*(2), 566–576.

Liu, F., Zhao, W., Chen, Y., Wang, Z., Yang, T., & Jiang, L. (2021). Sstdp: Supervised spike timing dependent plasticity for efficient spiking neural network training. *Frontiers in Neuroscience*, *15*.

Lobov, S. A., Mikhaylov, A. N., Shamshin, M., Makarov, V. A., & Kazantsev, V. B. (2020). Spatial properties of stdp in a self-learning spiking neural network enable controlling a mobile robot. *Frontiers*

in neuroscience, *14*, 88.

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural networks*, *10*(9), 1659–1671.

Mahowald, M. (1994). The silicon retina. In *An analog vlsi system for stereoscopic vision* (pp. 4–65). Springer.

Maro, J.-M., Ieng, S.-H., & Benosman, R. (2020). Event-based gesture recognition with dynamic background suppression using smartphone computational capabilities. *Frontiers in neuroscience*, *14*, 275.

Martins, W. M., Braga, R. G., Ramos, A. C. B., & Mora-Camino, F. (2018). A computer vision based algorithm for obstacle avoidance. In *Information technology-new generations* (pp. 569–575). Springer.

Mead, C. (1990). Neuromorphic electronic systems. *Proceedings of the IEEE*, *78*(10), 1629–1636.

Mead, C. (2020). How we created neuromorphic engineering. *Nature Electronics*, *3*(7), 434–435.

Michaelis, C., Lehr, A. B., & Tetzlaff, C. (2020). Robust trajectory generation for robotic control on the neuromorphic research chip loihi. *Frontiers in neurorobotics*, *14*, 589532.

Milde, M. B., Bertrand, O. J., Benosmanz, R., Egelhaaf, M., & Chicca, E. (2015). Bioinspired event-driven collision avoidance algorithm based on optic flow. In *2015 international conference on event-based control, communication, and signal processing (ebccsp)* (pp. 1–7).

Milde, M. B., Blum, H., Dietmüller, A., Sumislawska, D., Conradt, J., Indiveri, G., & Sandamirskaya, Y. (2017). Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system. *Frontiers in neurorobotics*, *11*, 28.

Minguez, J., Lamiraux, F., & Laumond, J.-P. (2016). Motion planning and obstacle avoidance. In *Springer handbook of robotics* (pp. 1177–1202). Springer.

Mirsadeghi, M., Shalchian, M., Kheradpisheh, S. R., & Masquelier, T. (2021). Stidi-bp: Spike time displacement based error backpropagation in multilayer spiking neural networks. *Neurocomputing*, *427*, 131–140.

Mitrokhin, A., Fermüller, C., Parameshwara, C., & Aloimonos, Y. (2018). Event-based moving object detection and tracking. In *2018 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 1–9).

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE transactions on neural networks and learning systems*, *29*(7), 3227–3235.

Mronga, D., Knobloch, T., de Gea Fernández, J., & Kirchner, F. (2020). A constraint-based approach for human–robot collision avoidance. *Advanced Robotics*, *34*(5), 265–281.

Neftci, E. O., Mostafa, H., & Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, *36*(6), 51–63.

Neil, D., Pfeiffer, M., & Liu, S.-C. (2016). Learning to be efficient: Algorithms for training low-latency, low-compute deep spiking neural networks. In *Proceedings of the 31st annual acm symposium on applied computing* (pp. 293–298).

Park, D.-H., Hoffmann, H., Pastor, P., & Schaal, S. (2008). Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *Humanoids 2008-8th ieee-ras international conference on humanoid robots* (pp. 91–98).

Pfeiffer, M., & Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Frontiers in neuroscience*, 774.

Ponghiran, W., Srinivasan, G., & Roy, K. (2019). Reinforcement learning with low-complexity liquid state machines. *Frontiers in Neuroscience*, *13*, 883.

Posch, C., Matolin, D., & Wohlgenannt, R. (2010). A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds. *IEEE Journal of Solid-State Circuits*, *46*(1), 259–275.

Posch, C., Serrano-Gotarredona, T., Linares-Barranco, B., & Delbruck, T. (2014). Retinomorphic event-based vision sensors: bioinspired cameras with spiking output. *Proceedings of the IEEE*, *102*(10), 1470–1484.

Rai, A., Meier, F., Ijspeert, A., & Schaal, S. (2014). Learning coupling terms for obstacle avoidance. In *2014 ieee-ras international conference on humanoid robots* (pp. 512–518).

Rajendran, B., Sebastian, A., Schmuker, M., Srinivasa, N., & Eleftheriou, E. (2019). Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches. *IEEE Signal Processing Magazine*, *36*(6), 97–110.

Rebecq, H., Gehrig, D., & Scaramuzza, D. (2018). Esim: an open event camera simulator. In *Conference on robot learning* (pp. 969–982).

Rebecq, H., Ranftl, R., Koltun, V., & Scaramuzza, D. (2019). High speed and high dynamic range video with an event camera. *IEEE transactions on pattern analysis and machine intelligence*, *43*(6), 1964–1980.

Reiter, P., Jose, G. R., Bizmpikis, S., & Cîrjilă, I.-A. (2020). Neuromorphic processing and sensing: Evolutionary progression of ai to spiking. *arXiv preprint arXiv:2007.05606*.

Risi, N., Aimar, A., Donati, E., Solinas, S., & Indiveri, G. (2020). A spike-based neuromorphic architecture of stereo vision. *Frontiers in neurorobotics*, 93.

Roy, K., Jaiswal, A., & Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature*, *575*(7784), 607–617.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Safeea, M., Neto, P., & Bearee, R. (2019). On-line collision avoidance for collaborative robot manipulators by adjusting off-line generated paths: An industrial use case. *Robotics and Autonomous Systems*, *119*, 278–288.

Salarpour, A., & Khotanlou, H. (2019). Direction-based similarity measure to trajectory clustering. *IET Signal Processing*, *13*(1), 70–76.

Salvatore, N., Mian, S., Abidi, C., & George, A. D. (2020). A neuro-inspired approach to intelligent collision avoidance and navigation. In *2020 aiaa/ieee 39th digital avionics systems conference (dasc)* (pp. 1–9).

Sanket, N. J., Parameshwara, C. M., Singh, C. D., Kuruttukulam, A. V., Fermüller, C., Scaramuzza, D., & Aloimonos, Y. (2020). Evdodgenet: Deep dynamic obstacle dodging with event cameras. In *2020 ieee international conference on robotics and automation (icra)* (pp. 10651–10657).

Schaub, A., Baumgartner, D., & Burschka, D. (2016). Reactive obstacle avoidance for highly maneuverable vehicles based on a two-stage optical flow clustering. *IEEE Transactions on Intelligent Transportation Systems*, *18*(8), 2137–2152.

Scheerlinck, C., Rebecq, H., Gehrig, D., Barnes, N., Mahony, R., & Scaramuzza, D. (2020). Fast image reconstruction with an event camera. In *Proceedings of the ieee/cvf winter conference on applications of computer vision* (pp. 156–163).

Scoccia, C., Palmieri, G., Palpacelli, M. C., & Callegari, M. (2021). A collision avoidance strategy for redundant manipulators in dynamically variable environments: on-line perturbations of off-line generated trajectories. *Machines*, *9*(2), 30.

Serre, T. (2019). Deep learning: the good, the bad, and the ugly. *Annual review of vision science*, *5*(1), 399–426.

Shrestha, S. B., & Orchard, G. (2018). Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems*, *31*.

Simon, D., & Isik, C. (1993). A trigonometric trajectory generator for robotic arms. *International Journal of Control*, *57*(3), 505–517.

Song, K.-T., Chang, Y.-H., & Chen, J.-H. (2019). 3d vision for object grasp and obstacle avoidance of a collaborative robot. In *2019 ieee/asme international conference on advanced intelligent mechatronics (aim)* (pp. 254–258).

Stagsted, R., Vitale, A., Binz, J., Bonde Larsen, L., Sandamirskaya, Y., et al. (2020). Towards neuromorphic control: A spiking neural network based pid controller for uav..

Stuijt, J., Sifalakis, M., Yousefzadeh, A., & Corradi, F. (2021). μbrain: An event-driven and fully synthesizable architecture for spiking neural networks. *Frontiers in neuroscience*, *15*, 538.

Sun, S., Cioffi, G., De Visser, C., & Scaramuzza, D. (2021). Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events. *IEEE Robotics and Automation Letters*, *6*(2), 580–587.

Sutton, R. S., Barto, A. G., & Others. (1998). *Introduction to reinforcement learning* (Vol. 135). MIT press Cambridge.

Taherkhani, A., Belatreche, A., Li, Y., Cosma, G., Maguire, L. P., & McGinnity, T. M. (2020). A review of learning in biologically plausible spiking neural networks. *Neural Networks*, *122*, 253–272.

Tang, G., Kumar, N., Yoo, R., & Michmizos, K. P. (2020). Deep reinforcement learning with population-coded spiking neural network for continuous control. *arXiv preprint arXiv:2010.09635*.

Taunyazov, T., Sng, W., See, H. H., Lim, B., Kuan, J., Ansari, A. F., ... Soh, H. (2020). Event-driven visual-tactile sensing and learning for robots. *arXiv preprint arXiv:2009.07083*.

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. (2019). Deep learning in spiking neural networks. *Neural networks*, *111*, 47–63.

Thakur, C. S., Molin, J. L., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., ... others (2018).

Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in neuroscience*, 891.

Tuckwell, H. C., & Wan, F. Y. (2005). Time to first spike in stochastic hodgkin–huxley systems. *Physica A: Statistical Mechanics and its Applications*, *351*(2-4), 427–438.

Tulbure, A., & Khatib, O. (2020). Closing the loop: Real-time perception and control for robust collision avoidance with occluded obstacles. In *2020 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 5700–5707).

Van Der Smagt, P., Arbib, M. A., & Metta, G. (2016). Neurorobotics: From vision to action. In *Springer handbook of robotics* (pp. 2069–2094). Springer.

Vitale, A., Renner, A., Nauer, C., Scaramuzza, D., & Sandamirskaya, Y. (2021). Event-driven vision and control for uavs on a neuromorphic chip. In *2021 ieee international conference on robotics and automation (icra)* (pp. 103–109).

Wang, X., Lin, X., & Dang, X. (2020). Supervised learning in spiking neural networks: A review of algorithms and evaluations. *Neural Networks*, *125*, 258–280.

Wicaksono, D. H. (2008). *Learning from nature: biologically-inspired sensors*. TU Delft, Delft University of Technology.

Wunderlich, T., Kungl, A. F., Müller, E., Hartel, A., Stradmann, Y., Aamir, S. A., ... others (2019). Demonstrating advantages of neuromorphic computation: a pilot study. *Frontiers in neuroscience*, *13*, 260.

Yasin, J. N., Mohamed, S. A., Haghbayan, M.-h., Heikkonen, J., Tenhunen, H., Yasin, M. M., & Plosila, J. (2020). Night vision obstacle detection and avoidance based on bio-inspired vision sensors. In *2020 ieee sensors* (pp. 1–4).

Zahra, O., Tolu, S., & Navarro-Alarcon, D. (2021). Differential mapping spiking neural network for sensor-based robot control. *Bioinspiration & Biomimetics*, *16*(3), 036008.

Zenke, F., Bohté, S. M., Clopath, C., Comşa, I. M., Göltz, J., Maass, W., ... others (2021). Visualizing a joint future of neuroscience and neuromorphic engineering. *Neuron*, *109*(4), 571–575.

Zhang, H., Jin, H., Liu, Z., Liu, Y., Zhu, Y., & Zhao, J. (2019). Real-time kinematic control for redundant manipulators in a time-varying environment: Multiple-dynamic obstacle avoidance and fast tracking of a moving object. *IEEE Transactions on Industrial Informatics*, *16*(1), 28–41.

Zheng, H., Wu, Y., Deng, L., Hu, Y., & Li, G. (2020). Going deeper with directly-trained larger spiking neural networks. *arXiv preprint arXiv:2011.05280*.

Zhou, S., Wang, W., Li, X., & Jin, Z. (2021). A spike learning system for event-driven object recognition. *arXiv preprint arXiv:2101.08850*.

Zhou, Y., Gallego, G., & Shen, S. (2021). Event-based stereo visual odometry. *IEEE Transactions on Robotics*, *37*(5), 1433–1450.