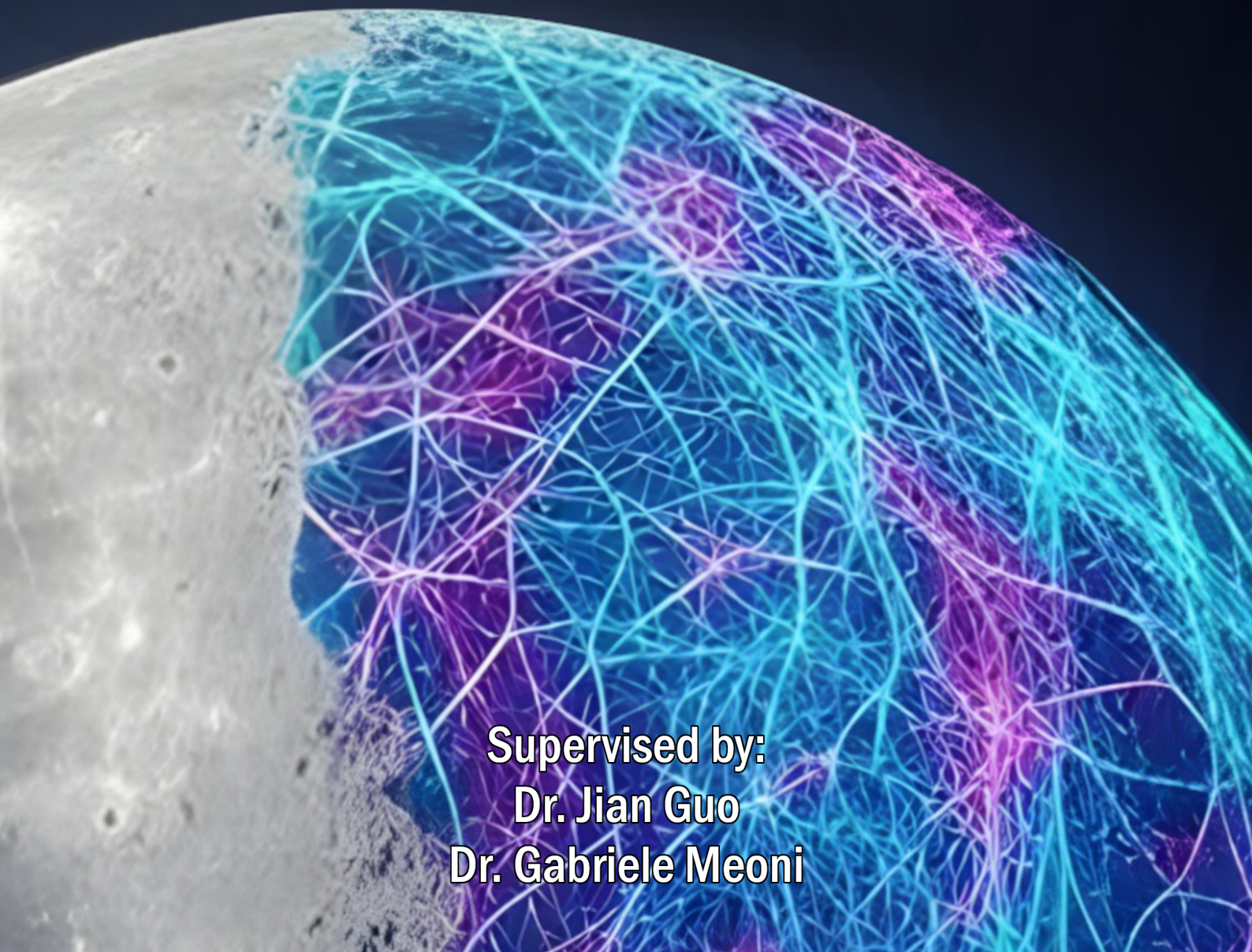
TU Delft

# Lunar Landing Navigation with an Event Camera and Spiking Neural Networks

Ondřej Dvořák



Supervised by:
Dr. Jian Guo
Dr. Gabriele Meoni

TU Delft

# MSc Thesis - Space Engineering

## Lunar Landing Navigation with an Event Camera and Spiking Neural Networks

by

**Ondřej Dvořák**

**Supervised by:**
Dr. Jian Guo
Dr. Gabriele Meoni

# Acknowledgments

This work marks the conclusion of my six years in the beautiful city of Delft. These were not easy years, yet they have been the best years of my life so far. I would like to express my gratitude to everyone who helped me bring this work to completion.

First and foremost, I would like to thank my supervisors, Professor Jian Guo from TU Delft and Dr. Gabriele Meoni from ESA, for guiding me through this incredibly complex yet rewarding topic. I am also deeply grateful to Dr. Dario Izzo from ESA for his invaluable advice.

I would also like to thank Adam, who spent weeks with me in a small, dark laboratory on top of the faculty building in the middle of a warm summer. Without him, this work would have been an extremely lonely endeavor.

Thirdly, I want to thank my family for their unwavering support throughout this journey. I could not have completed this work without them.

Lastly, I am especially grateful to Maria for keeping my spirits high through all the low points of this research and for giving me the motivation to finish it. It was an almost impossible task, as there were many great obstacles, yet she somehow always managed.

# Contents

**List of Abbreviations** **6**

**List of Symbols** **7**

**List of Figures** **7**

**List of Tables** **9**

**1 Introduction** **11**
1.1 Opening remarks . . . 11
1.2 Key concepts . . . 11
1.2.1 Lander trajectory optimization . . . 11
1.2.2 Event camera . . . 12
1.2.3 Spiking neural network . . . 13
1.2.4 Optical flow . . . 13
1.3 Research questions . . . 13
1.4 Process overview . . . 16

**I Event-based landing dataset** **17**

**2 Optimal trajectories** **19**
2.1 Coordinate system and the optimal landing problem . . . 19
2.1.1 Coordinate system . . . 19
2.1.2 Optimal landing problem . . . 20
2.2 Apollo lander model and its trajectories . . . 21
2.2.1 Mass and inertia . . . 21
2.2.2 Main engine and thrusters . . . 21
2.2.3 Apollo trajectories . . . 21
2.3 Choice of the optimization method . . . 22
2.3.1 Description of G-FOLD . . . 22
2.3.2 Comparison of G-FOLD and general non-linear programming . . . 22
2.4 Implementation of the optimization method . . . 23
2.4.1 Problem transcription and solver . . . 23
2.4.2 Optimal control problem initialization . . . 23
2.5 Trajectory types . . . 24
2.5.1 Ventral trajectories . . . 24
2.5.2 Nominal trajectories . . . 24
2.5.3 Divert trajectories . . . 24
2.6 Resultant trajectories . . . 24
2.6.1 Trajectory initial conditions . . . 25
2.6.2 Trajectory overview . . . 25

**3 Surface model and ground truth motion fields** **27**
3.1 Pinhole camera . . . 27
3.2 Landing site and lighting conditions . . . 28
3.2.1 Landing site . . . 28
3.2.2 Lighting conditions . . . 28
3.3 Rendering of lunar scenery . . . 29
3.3.1 Surface model . . . 29

3.3.2 Frame interpolation . . . 29
3.4 Ground truth motion fields . . . 30
3.4.1 Estimating motion fields . . . 30
3.4.2 Generating depth maps . . . 30

**4 Event camera simulation and model validation** **31**
4.1 Event camera simulator testing . . . 31
4.1.1 Test method . . . 31
4.1.2 Computational cost . . . 32
4.1.3 Timestamp fidelity . . . 32
4.1.4 Event count and distribution . . . 33
4.1.5 Ease-of-use and flexibility . . . 34
4.1.6 Trade-off between event camera simulators . . . 34
4.2 Event camera model . . . 34
4.2.1 Simulated non-ideal behaviors . . . 35
4.2.2 Model limitations . . . 35
4.3 Validation procedure . . . 35
4.3.1 Choice of ground truth data . . . 35
4.3.2 Validation metrics . . . 36
4.3.3 Processing of event data . . . 37
4.3.4 Tuning and results . . . 37

**II Learning optical flow and estimating egomotion** **40**

**5 Learning optical flow using spiking neural networks** **42**
5.1 Choice of the SNN models for evaluation . . . 42
5.2 Network architecture and training . . . 42
5.3 Handling of temporal event information . . . 43
5.3.1 Network input . . . 44
5.3.2 Neuron models and temporal convolution . . . 44
5.4 Loss functions . . . 45
5.5 Benchmark . . . 46
5.6 Experiments and results . . . 46
5.6.1 Implementation . . . 46
5.6.2 Method of training . . . 47
5.6.3 Average Endpoint Error . . . 47
5.6.4 Spatial error correlation . . . 47

**6 Optical flow to egomotion** **50**
6.1 Problem overview . . . 50
6.1.1 Problem introduction . . . 50
6.1.2 Information regimes . . . 51
6.2 Solving the overdetermined system for egomotion . . . 53
6.2.1 Known Altitude and Attitude regime . . . 53
6.2.2 Rangefinder regime . . . 53
6.2.3 Known Altitude regime . . . 55
6.3 Error propagation and sensitivity to spatially correlated errors . . . 55
6.3.1 Error propagation and sensitivity . . . 56
6.3.2 Spatially correlated errors . . . 57
6.4 Navigational filter . . . 57

**7 Results** **60**
7.1 Altitude-Corrected Egomotion Errors . . . 60
7.1.1 Altitude independence . . . 60
7.1.2 Performance in different regimes . . . 61
7.2 Relative egomotion error . . . 63

**III Final remarks** **64**

**8 Conclusion** **65**
8.1 Answers to research questions . . . 65
8.2 Recommendations for future work . . . 67

**A Event camera and event-based datasets** **69**
A.1 Biological retinas . . . 69
A.1.1 Structure of biological retinas . . . 69
A.1.2 Features of biological retinas . . . 69
A.2 Event camera principles and advantages . . . 70
A.2.1 Differences between an event camera and a frame-based camera . . . 70
A.2.2 Principles of event cameras . . . 71
A.3 Review of currently available event cameras . . . 71
A.4 Strategies for generating event-based datasets . . . 73
A.4.1 Planetary landing event-based dataset by use of physical camera . . . 73
A.4.2 Planetary landing event-based dataset by a conversion from a frame-based dataset . . . 74
A.5 Event camera simulators . . . 74
A.5.1 Pixel behavior . . . 74
A.5.2 Deterministic simulators . . . 75
A.5.3 Stochastic simulators . . . 76
A.5.4 Learning-based simulators . . . 76

**B Trajectories for landing on an airless body** **77**
B.1 Lunar landing . . . 77
B.1.1 Lunar lander vehicles . . . 77
B.1.2 Landing trajectories . . . 77
B.2 Trajectory optimization . . . 79
B.3 Modern methods for fuel-optimal landing . . . 79
B.3.1 Non-linear programming . . . 79
B.3.2 Convex optimization . . . 80
B.3.3 General non-linear programming . . . 80

**C Artificial neural networks in the context of event-based optical flow** **81**
C.1 Artificial neural networks . . . 81
C.1.1 Basic principles of artificial neural networks . . . 81
C.1.2 Backpropagation . . . 82
C.2 Spiking neural networks . . . 82
C.2.1 Biological background . . . 82
C.2.2 Spiking neural network principles . . . 83
C.2.3 Neuron models for spiking neural networks . . . 83
C.2.4 Learning in spiking neural networks . . . 85
C.2.5 Neuromorphic hardware . . . 86
C.2.6 Simulation of spiking neural networks . . . 87
C.3 Convolutional neural networks . . . 87
C.3.1 Convolutional neural network principles . . . 87

**D Optical flow and learning-based methods for optical flow estimation from events** **89**
D.1 Principles of optical flow . . . 89
D.1.1 Optical flow and motion field . . . 89
D.1.2 Aperture problem . . . 90
D.2 Spiking neural networks using synaptic delays . . . 91
D.2.1 Spiking Architecture for Visual Motion Estimation . . . 91
D.2.2 Convolutional neural networks with delay blocks . . . 92
D.3 U-Net-like convolutional neural networks . . . 92
D.3.1 Analog U-Nets . . . 93
D.3.2 Spiking and analog hybrid U-Nets . . . 94
D.3.3 Fully spiking U-Nets . . . 94
D.4 Other spiking architectures . . . 95
D.5 Other analog architectures . . . 96

**E Additional results** **97**
E.1 Dataset update . . . . . 97
E.2 Model retraining . . . . . 97

# List of Abbreviations

| Abbreviation | Description |
|---|---|
| AC | Anti-Correlation loss term |
| ACEE | Altitude Corrected Egomotion Error |
| AEE | Average Endpoint Error |
| AER | Adress Event Representation |
| ASN | Adaptive-SpikeNet |
| BM | Benchmark |
| DEM | Digital Elevation Model |
| DSEC | Stereo Event Camera Dataset for Driving Scenarios |
| EE | Egomotion Error |
| IF | Integrate-and-Fire |
| KAA | Known Altitude and Attitude regime |
| KA | Known Altitude regime |
| LIF | Leaky Integrate-and-Fire |
| MVSEC | Multi Vehicle Stereo Event Camera Dataset |
| OES | OF-EV-SNN |
| PANGU | Planet and Asteroid Natural scene Generation Utility |
| R | Rangefinder regime |
| RCS | Reaction Control System |
| REE | Relative Egomotion Error |
| SLT | Synthetic Lunar Terrain dataset |
| SFN | Spike-FlowNet |
| SNN | Spiking Neural Network |

# List of Symbols

| Symbol | Description | Unit |
|---|---|---|
| $F_a$ | Main engine max thrust | $[N]$ |
| $\bar{F}_b$ | RCS max thrust | $[N]$ |
| $\boldsymbol{\omega_b}$ | Angular velocity in the body reference frame | $[rad/s]$ |
| $\boldsymbol{\omega_b}$ | Angular velocity in the inertial reference frame | $[rad/s]$ |
| $\mathcal{F}_b$ | Body reference frame | – |
| $\mathcal{F}_i$ | Inertial reference frame | – |
| $\mathcal{L}_a$ | Average Endpoint Error loss | $[px]$ |
| $\mathcal{L}_b$ | Anti-correlation loss term | $[px]$ |
| $\mathbf{H}$ | Homography matrix | $[m^2/s]$ |
| $\mathbf{I}$ | Vehicle moment of inertia | $[kgm^2]$ |
| $\mathbf{W}$ | Vector of synaptic weights | $[-]$ |
| $\mathbf{h}$ | Homography vector | $[m^2/s]$ |
| $\mathbf{n}$ | Normal unit vector to the lunar surface in the body reference frame | $[m/s]$ |
| $\mathbf{o}$ | Vector of spikes | $[-]$ |
| $\mathbf{r} = [x, y, z]^T$ | Position in the inertial reference frame | $[m]$ |
| $\mathbf{r_b} = [X, Y, Z]^T$ | Position in the body reference frame | $[m]$ |
| $\mathbf{u}$ | Vector of neuron potentials | $[V]$ |
| $\mathbf{v}$ | Velocity in inertial reference frame | $[m/s]$ |
| $\mathbf{v_b}$ | Velocity in body reference frame | $[m/s]$ |
| $\hat{\mathbf{u}}$ | Optical flow / Motion field | $[px/s]$ |
| $\hat{\mathbf{u}}'$ | Derotated optical flow / Derotated motion field | $[px/s]$ |
| $\mathbf{z}$ | Vector of spike thresholds | $[-]$ |
| $f$ | Focal length | $[m]$ |
| $I$ | Intensity | $[cd]$ |
| $k_e$ | Exhaust velocity | $[m/s]$ |
| $L$ | Distance between RCS thrusters | $[m]$ |
| $l_{rf}$ | Slanted range from rangefinder | $[m]$ |
| $l_s$ | Slanted range from snapshot LIDAR | $[m]$ |
| $m$ | Vehicle mass | $[kg]$ |
| $\phi$ | Roll angle | $[rad]$ |
| $\psi$ | Yaw angle | $[rad]$ |
| $R$ | Ground distance | $[m]$ |
| $\rho$ | Anti-correlation loss term weight | $[-]$ |
| $\sigma_\theta$ | Threshold mismatch | $[-]$ |
| $t$ | Time | $[s]$ |
| $\theta$ | Pitch angle | $[rad]$ |
| $\theta_{ON/OFF}$ | Event camera logarithmic threshold | $[-]$ |
| $u_t$ | Main engine thrust level | $[-]$ |
| $u_\phi, u_\theta, u_\psi$ | RCS thrust levels | $[-]$ |
| $v_{th}$ | Potential threshold | $[V]$ |
| $W$ | Image plane width/height | $[px]$ |
| $\hat{x}, \hat{y}$ | Image plane coordinates | $[px]$ |

# List of Figures


1.1 The principle of event camera . . . 12
1.2 Example of a U-Net architecture . . . 14
1.3 Illustration of different optical flow fields produced by different relative motions . . . 14
1.4 Main steps and the relevant chapters . . . 16
1.5 Overview of the event-based landing dataset generation and validation process . . . 18

2.1 The two reference frames used in the work . . . 19
2.2 Comparison between pitch angle profile generated by G-FOLD and a general non-linear program 23
2.3 Vertical and horizontal velocities on the dataset trajectories plotted against altitude. . . . 25
2.4 Angular velocities on the dataset trajectories plotted against altitude. . . . 26

3.1 Projection of a point in the non-inertial frame on the image plane . . . 27
3.2 Topographic map of the Sverdrup crater . . . 28
3.3 PANGU rendered regions with a different level of detail . . . 29

4.1 Point cloud samples generated by different event camera simulators . . . 33
4.2 Histograms of time interval between consecutive events on a pixel for the ground truth and select event camera simulators. . . . 34
4.3 Spikes in event count per input frame caused by the discrete level of detail changes in the underlying surface model . . . 36
4.4 Iterative tuning process . . . 38
4.5 Event rate and its variance during the validation process. . . . 39
4.6 Overview of the process of learning the optical flow and egomotion inference . . . 41

5.1 U-Net-like architecture of the chosen networks . . . 43
5.2 Distribution of AEE for the five different models and three different trajectory types. . . . 48
5.3 Average Endpoint Error as a function of average optical flow in OES AC and BM models. . . . 48
5.4 The distribution of Anti-Correlation term AC among the five models and three trajectory types. 49

6.1 The event camera field of view and the rangefinder. . . . 52
6.2 Unit normal vector $\mathbf{n}$, reference frame $\mathcal{F}_b$ and the relevant distances. . . . 53
6.3 Example of point sampling from a 200x200 square. . . . 56
6.4 Mean and one standard deviation of ACEE as a function of AEE as a result of Monte Carlo experiments for the three information regimes. . . . 57
6.5 Adverse effects of smooth optical flow field with spatially correlated angular errors illustrated by Monte Carlo experiments. . . . 58
6.6 Setup of the two low-pass filters used in the R and KA regime. . . . 58

7.1 ACEE plotted against the altitude $z$ for OES AC and BM models. . . . 60
7.2 ACEE inferred using the OES AC model plotted against the altitude $z$ for all the test trajectories. 61
7.3 ACEE inferred using the BM model plotted against the altitude $z$ for all the test trajectories. . . 62
7.4 Altitude-corrected velocity against altitude in the test dataset. . . . 63

A.1 Structure of a biological retina . . . 70
A.2 Comparison of event camera and a frame-based camera outputs . . . 70
A.3 Similarities between pixel circuitry of an event camera and a biological retina . . . 72
A.4 The Dynamic Vision Sensor of Lichtsteiner et al. (2008) . . . 72
A.5 Process of creation of event-based landing dataset by use of event camera mounted on a robotic arm . . . 73
A.6 Model of a pixel circuit for noise simulation . . . 74
A.7 Pixel model used by the v2e event camera simulator . . . 75

B.1 Simple diagram of the Surveyor lander . . . 78
B.2 Proposed phases of a lunar landing for the cancelled Altair vehicle . . . 78

C.1 Illustration of a simple neural networks structure . . . 81
C.2 Simple schema of a pre-synaptic and a post-synaptic biological neuron . . . 83
C.3 An illustration of coding schemes for input to spiking neural networks . . . 84
C.4 An illustration of a circuit of a leaky integrate-and-fire neuron . . . 85
C.5 Qualitative comparison of von Neumann architectures and neuromorphic architectures . . . 86
C.6 An illustration of a two-dimensional convolution kernel . . . 87
C.7 Illustration of the pooling mechanism . . . 88
C.8 Schematic diagram of a simple convolutional neural network architecture . . . 88

D.1 Demonstration of difference between optical flow and motion field . . . 90
D.2 The barbers pole illusion illustration . . . 90
D.3 Illustration of the aperture problem . . . 91
D.4 An illustration of SAVME principle . . . 91
D.5 Architecture of network proposed by Chaney et al. (2021) . . . 92
D.6 Architecture of the EV-FlowNet . . . 93
D.7 An illustration of Spike-FlowNet architecture . . . 94
D.8 An illustration architecture used by Cuadrado et al. (2023) . . . 95
D.9 SNN-FireNet architecture used by Hagenaars et al. (2021) . . . 96

E.1 Visual comparison between the original dataset and its update. . . . 97
E.2 Plot of average angular error in the optical flow against the angular velocity in updated ASN AC and OES AC models. . . . 98

# List of Tables


2.1 Apollo mass and inertia properties used in the simulations. . . . . 21
2.2 Apollo engine and RCS parameters used in the simulations. . . . . 21
2.3 Initial and final conditions for the test case to compare G-FOLD with a non-linear program . . . 22
2.4 Parameters used to generate major trajectories and minor trajectory start altitude . . . . . 25

4.1 Event camera simulators chosen for testing . . . . . 31
4.2 Frames processed per second for the different event camera simulators under testing. . . . . 32
4.3 Trade-off table assessing qualities of different event camera simulators. . . . . 35
4.4 Values for event camera simulator non-ideality parameters. . . . . 35

5.1 Average endpoint error performance comparison of various well-performing models on the indoor trajectories of the MVSEC dataset. . . . . 42
5.2 Mean AEE values for models across trajectory types (V, N, D) and their totals. . . . . 47

6.1 Known additional variables for each regime. . . . . 52

7.1 Mean values of ACEE and their standard deviations when using the OES AC model. . . . . 62
7.2 Mean values of ACEE and their standard deviations when using the BM model. . . . . 63
7.3 ACEE difference between OES AC and BM models in percent. . . . . 63

A.1 Parameters of DVS event camera by Lichtsteiner et al. (2008). . . . . 72

E.1 Mean AEE values for models across trajectory types (V, N, D) and their totals on the improved dataset . . . . . 98
E.2 Updated ACEE values using the OES AC model trained on the improved dataset. . . . . 98
E.3 ACEE difference between OES AC models before and after update in percent. . . . . 99

# Chapter 1

# Introduction

This chapter first presents a brief opening to the report in Section 1.1. Then, for convenience, four key concepts used in this work are presented in Section 1.2. It is recommended to study these concepts in full using the literature study provided in the Appendix, but if the reader wants to merely understand the rest of the report, the provided descriptions should suffice. Afterwards, research questions are introduced and motivated in Section 1.3. Lastly, a short overview of the workflow and relevant chapters is given in Section 1.4.

## 1.1 Opening remarks

After 50 years humanity is again very actively looking towards the Moon. Scientists, private companies, and government organizations are interested in exploring and exploiting the Moon from both the orbit and the surface. However, landing on the surface is an extremely challenging task from the perspective of guidance, navigation, and control. Only during the period of this research, four landing attempts occurred and just one of them went according to the plan. Especially the Nova-C lander by Intuitive Machines beautifully frames the context of this work. According to Foust (2024), the lander was left without a large part of its navigation hardware during its descent. However, the team managed to use a camera, which was originally meant as a payload, to obtain navigational information and saved the mission. This unplanned but ingenious redundancy saved an incredibly costly project from a large setback.

It is very desirable to build robust navigation systems that employ redundancy. To do so, we must continually strive to develop cheaper, lighter, smaller, and more efficient sensors. These efforts also allow us to build smaller landers such as OMOTENASHI from JAXA described by Hirasawa et al. (2025). One of the great prospective options in this domain are vision sensors and among them event camera. And this work focuses specifically on this bio-inspired device and its application in lunar landing. Coupled with a novel and also bio-inspired processing paradigm called the spiking neural network, which is very well suited for this match, it has a lot of potential for future missions.

And now some are looking in this direction. Among them is the Advanced Concepts Team of the European Space Agency, which intends to use their platform to organize a sizable competition focused on neuromorphic processing during lunar landings. Although technically separate, this work was executed in collaboration with the team and is to be used to provide a dataset and a baseline for the competition. Hence, it takes the form of a feasibility study with a focus on the realistic simulation of the sensors, and it is by no means an exhaustive comparison of all available methods. The purpose of this work is to show that this is an idea worth studying and to highlight some of the main challenges.

## 1.2 Key concepts

This section will briefly introduce four key concepts used in this work, which are necessary for understanding it - lander trajectory optimization using nonlinear programming, event cameras, spiking neural networks, and optical flow. As stated previously, more detailed information can be found in the literature study which is included in the Appendix.

### 1.2.1 Lander trajectory optimization

The work, which follows, focuses on the reconstruction of egomotion on fuel-optimal trajectories. In the modern era, these problems are generally solved using the so-called non-linear programming, as Malyuta, Yu, et al.

(2021) explain. As Luenberger (2003) states, non-linear programming involves solving problems of the following type with $m$ inequality constraints and $p$ equality constraints. Hence, the trajectory is first discretized, and then optimization methods are applied to it.

$$\begin{array}{ll} \text{minimize} & J(t) \\ \text{subject to} & f_i(t) \leq 0, \quad i = 1, \ldots, m \\ & h_i(t) = 0, \quad i = 1, \ldots, p \end{array} \tag{1.1}$$

However, the exact formulation of the optimization problem can have a major impact on the complexity of the solution. Some simple formulations will involve minimizing convex functions over convex regions and thus can be solved using convex optimization. This is of great benefit because as Malyuta, Reynolds, et al. (2021) explain, such algorithms are bound to find the global optimum in polynomial time. On the other hand, formulations involving rotational dynamics or other more advanced cases will typically require the use of general non-linear program solvers.

### 1.2.2 Event camera

Event camera is a vision sensor that asynchronously outputs the so-called events based on the change of brightness in the individual independent pixels, as explained by Lichtsteiner et al. (2008). It is loosely based on the function of human retina, which makes it the so-called neuromorphic device.

The event will be produced on a pixel every time the camera registers a change in logarithm of intensity $I$ larger than a predefined threshold $\theta_{ON/OFF}$, as seen in the expression below. Events have only two polarities: if the pixel suddenly becomes brighter, a positive ON event is generated, and vice versa, a decrease in light will lead to a negative OFF event. After an event is produced, the reference intensity is reset on the pixel. The function of the pixel is illustrated in Figure 1.1.

$$\Delta log(I) \geq \theta_{ON/OFF} \tag{1.2}$$

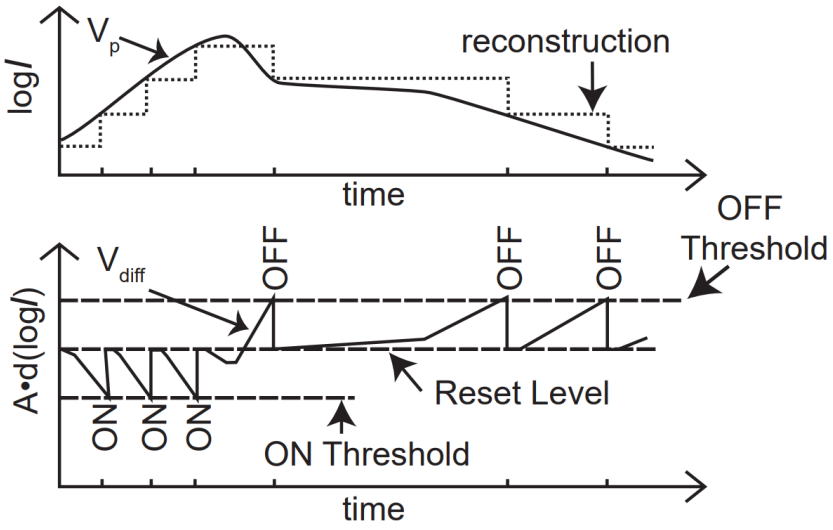



Figure 1.1: The principle of event camera. An ON or OFF event is produced in a certain pixel when the logarithm of intensity change reaches a certain threshold. Afterwards the reference intensity is reset and the cycle starts again. Image taken from Lichtsteiner et al. (2008).

Event camera is very well suited for space applications because of its low mass and power consumption, high dynamic range allowing for use with scenes containing extreme brightness and darkness, and sparse output as pointed out by Gallego et al. (2019).

However, there is a lack of lunar landing-relevant event-based datasets and it was deemed very difficult to attempt to generate one using a real event camera due to time and resource constraints. Hence, an event camera simulator must be used. Current state-of-the-art event camera simulators like, e.g. the one of Hu et al. (2020), convert frame-based video into an event stream.

Apart from the rather straightforward change in brightness, event camera simulators also typically focus on a number of non-idealities characteristic for the instrument. Perhaps the most important is the low-pass filter-like behavior of the camera pixels described by Lichtsteiner et al. (2008). The values of thresholds $\theta_{ON}$ and $\theta_{OFF}$ are actually not constant across the image plane due to the effect called transistor mismatch, as stated in Lichtsteiner et al. (2008).

Moreover, to determine the change in brightness, the camera pixel memorizes the brightness value at its last event fired. However, as explained by Hu et al. (2020), this memorized value decreases over time due to a non-ideal behavior called leakage current.

### 1.2.3 Spiking neural network

The event camera is complemented by another neuromorphic device called the spiking neural network. It is a novel computing paradigm, which can be implemented on very lightweight hardware known as neuromorphic chip, which also has low power consumption and latency, as explained by Schuman et al. (2022).

Unlike the common analog neural networks, which are described in the Appendix C and will not be treated here, the spiking neural networks incorporate the concept of time into the inference process. As W. Guo et al. (2021) describe, networks consist of neurons that communicate with each other using discrete time voltage spikes.

Incoming spikes increase the internal voltage of a neuron. Eventually, this internal voltage reaches a certain threshold and the neuron will fire an output spike itself. Then, the internal voltage decreases again. Commonly, the voltage builds up on the neuron over time until a spike (Integrate-and-Fire neurons), or it automatically decreases when there are no spikes incoming, thus prioritizing recent inputs (Leaky Integrate-and-Fire neurons).

Practical state-of-the-art spiking neural networks for optical flow determination typically use the convolutional neural network paradigm. Again, the reader is referred to the Appendix C in case of interest in the introduction to the topic. Specifically, networks of the so-called U-Net type introduced by Ronneberger et al. (2015) are particularly successful in this domain.

U-Net is an architecture with a contracting encoder and an expanding decoder, as illustrated in Figure 1.2. In the encoder, the input is downsized to a lower resolution by convolution to shift focus on global context. Then again, the output is up-sized to its original resolution in the decoder. Along this contraction/expansion path, some low-level detail can be lost. Hence, U-Net features the so-called skip connections to help ensure that this detail can propagate straight from early encoder layers to late layers in the decoder and it is preserved.

### 1.2.4 Optical flow

For this work, the concept of optical flow is very important. As defined by Horn and Schunck (1981), optical flow is *"the distribution of apparent velocities of movement of brightness patterns in an image"*. Another, perhaps more intuitive, explanation by Gibson and Marques (2016) states that it is projection of motion from a three-dimensional scene to a two-dimensional image plane.

Perceived optical flow is commonly used by humans, animals and insects, for navigation as pointed out in Warren et al. (2001). The optical flow vectors encode information about relative motion with regards to objects in the field of view. This is well illustrated by Figure 1.3.

In a similar fashion, optical flow can be utilized for lander navigation. Using the so-called continuous homography, which is described in detail in Chapter 6, it is possible to reconstruct both the lander pose and unscaled motion parameters. However, as Grabe et al. (2015) note, there is always an inherent scale ambiguity in this estimate. It is impossible to recognize using a monocular camera whether a fast-moving distant surface is being observed or a closer one moving at slower speed. Hence, optical flow data need to be combined with other sources like the rangefinder or altimeter which give absolute distance to the surface in order to obtain the velocity at the right scale.

## 1.3 Research questions

This section will introduce and motivate the research questions. Please note that this is a synthesis of the literature study which is presented in Appendix. For further detail, refer to the relevant chapters.

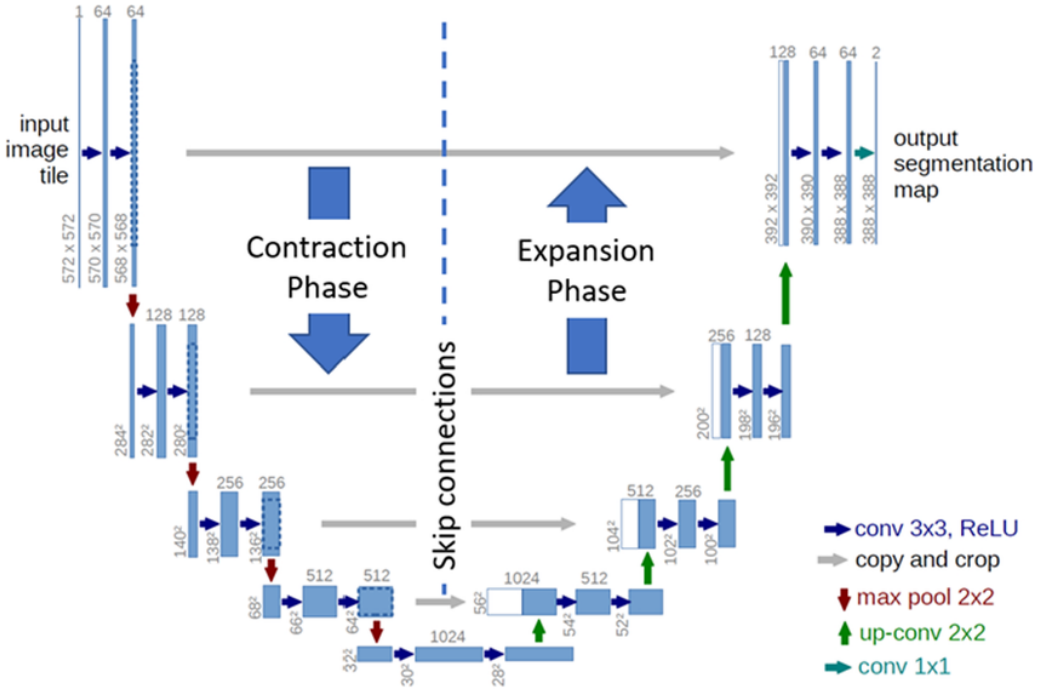



Figure 1.2: Example of U-Net architecture. The contracting path of the encoder progressively extracts features of higher and higher level. Fully connected layers in the middle increase the expressivity of the network. The expanding path of the decoder scales the output back up to the desired resolution. The skip connections ensure propagation of low-level details into the output. (Image source: Ronneberger et al. (2015))

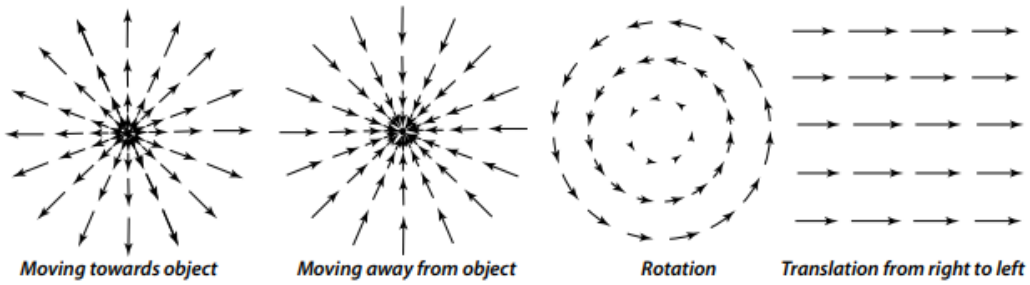



Figure 1.3: Illustration of different optical flow fields produced by different relative motions (image source: Distante and Distante (2020)).

With the lunar landings again becoming a very relevant topic – to the author's knowledge, at least six landing missions are planned for 2025 – further progress in area of lander navigation is required. The current common solutions, such as laser and radar velocimeters, are often rather bulky and hard to integrate into smaller vehicles and too heavy for a redundant system. That is why small landers like the Smart Lander for Investigating Moon began utilizing visual navigation using frame-based cameras as described by Ishida et al. (2025). A question is posed, whether it is possible to go one step further, and utilize even less demanding systems using novel vision sensors.

The event camera is a very light and low power device with low latency and high dynamic range as pointed out by Gallego et al. (2019). All of these are useful properties making it a good candidate for a vision sensor during lunar landing. Both use as a redundant device or a primary means of navigation on a smaller lander are open. This area was previously described and partially explored, for example by Izzo and Croon (2012) or McLeod et al. (2022), but there is no investigation involving lunar landings with complex trajectories. Considering that almost all landings on the Moon involve such trajectories, this is a fairly significant research gap. This topic is further explored in Appendix A.

Another device with extremely low latency, power consumption and mass is the neuromorphic chip capable of running spiking neural networks. While popular in the area of drone navigation, the author is unaware of any use of such networks in the context of combination of event-based inputs with planetary landings. The very

natural eye-and-brain-like compound system of event camera with spiking neural network is to be the topic of this research and frames the main research question. More information is provided in Appendix C.

**RQ** - How accurately can vehicle egomotion be reconstructed during a fuel-optimal planetary landing using an event camera output processed with a spiking neural network?

Such a study ultimately involves three different parts. The first one will focus on creating an event-based landing dataset, the second one will concentrate on training the spiking neural networks to output relevant data and the third one will come to processing the spiking neural network outputs.

The process of creating the event-based landing dataset is left open as a question (RSQ-1) and the following gaps are identified. While Azzalini et al. (2023) provide a good starting point in the area of dataset creation, there are multiple event camera simulators in contention including work by Zhu, Yuan, et al. (2018), Lin et al. (2022) and Z. Zhang et al. (2023). A question is asked, which one is best suited for the purpose (RSQ-1.1). Secondly, it is recognized that there is a lack of clear validation metrics for event datasets as pointed out by Lin et al. (2022), and the question of how such a dataset can be validated is posed (RSQ-1.2). Thirdly, as previously mentioned, optimal trajectories for the dataset can be generated using more complex general non-linear programming or using its simpler convex subset, as Malyuta, Yu, et al. (2021) point out. A question is raised whether the trajectories created using convex methods can still be deemed appropriate (RSQ-1.3). For further information, please refer to Appendix A and B.

**RSQ-1** - How can a synthetic event-based dataset for planetary landing be created?

- **RSQ-1.1** - What is a suitable event camera simulator in this application?
- **RSQ-1.2** - To what extent can such a dataset be validated?
- **RSQ-1.3** - How can the appropriate trajectories for the purpose be generated?

Spiking neural networks are found to be well suited for the purpose of prediction of optical flow, and there are a number of works focusing on this topic such as those of Hagenaars et al. (2021), Cuadrado et al. (2023), Schnider et al. (2023) or Kosta and Roy (2023). The decent results reported, together with the availability of such networks for testing, led to the adoption of optical flow as the desired spiking neural network output.

It is recognized that the problem of estimating optical flow during lunar landing differs from general optical flow estimation as the flow fields have a very special structure created by motion over nearly planar surface, constant lighting, and lack of moving objects. Hence, a focus is given on the difference between a general purpose network and a network for purposes of planetary landing (RSQ-2).

Specifically, a large number of different timing information processing methods were found ranging from explicit temporal convolution and recurrent units (Cuadrado et al. (2023), Hagenaars et al. (2021)) to reliance on inherent qualities of spiking neurons (Schnider et al. (2023), Kosta and Roy (2023)). Hence a question is asked, which of these methods is specifically well suited for the lunar landing (RSQ-2.1). For further elaboration on the topic of optical flow spiking neural networks please visit Appendix D.

Similarly, loss functions may need to take a special form and there is no consensus on what is the optimum even for the known datasets with various novel terms being commonly proposed as e.g. in Cuadrado et al. (2023). Hence, it is studied how should the loss function look in this case (RSQ-2.2).

**RSQ-2** - What is a good architecture of spiking neural network for processing the event stream during planetary landing?

- **RSQ-2.1** - How can the event timing information be well utilized in the spiking neural network?
- **RSQ-2.2** - What are the appropriate loss functions in the context of supervised learning of event-based optical flow?

The adequate step between optical flow and egomotion is left to be found during the research (RSQ-3). In particular, the flow is expected to be noisy, hence focus is given on appropriate filtering (RSQ-3.1). A question is asked in what conditions can useful egomotion estimates be derived from the optical flow (RSQ-3.2). Lastly, a decision is made to find out whether additional information, such as the pose of the lander, can significantly improve the egomotion estimates and what information should be provided to resolve the scale ambiguity mentioned in 1.2 (RSQ-3.3). This is an important question, as it may turn out that the event camera and spiking neural network are most useful in combination with another instrument. Information about egomotion recovery are not provided in the Appendix, but the current methods are detailed in Chapter 6.

**RSQ-3** - How can the egomotion be reconstructed from the output of a spiking neural network?

- **RSQ-3.1** - How should filtering be employed in order to increase performance?
- **RSQ-3.2** - In what conditions can useful egomotion estimates be generated from the optical flow?
- **RSQ-3.3** - How do the additional provided navigational information improve the egomotion estimates?

## 1.4 Process overview

This section is supposed to give the reader a general overview of the high-level methodology and steps involved in this research.

As already stated, the work is a feasibility study. As such, it aims to explore all the main aspects of egomotion estimation in rough terms and seek deeper understanding of the problem rather than centering on perfecting the solutions.

For this purpose, high-quality input data is vital. The first part of the work concerns itself with simulating the output of an event camera during an optimal lunar descent. This is a non-trivial task which consists of first generating the relevant frame-based videos and converting them into event streams using a simulator.

A spiking neural network can be trained using the available events to output optical flow. Two candidate architectures, one by Cuadrado et al. (2023) and one by Kosta and Roy (2023), are tested and trained. The architectures differ significantly from each other, especially in handling event timing information. The focus is also on appropriate loss functions. The aim here is to illustrate the common issues which may appear during this process.

The optical flow can be used to obtain motion estimates. That is, as already mentioned in Section 1.2, when it is combined with an output of another measurement instrument such as a rangefinder or an altimeter. It is investigated how good egomotion estimates can be obtained from the optical flow using such methods and how they vary based on the additional provided information like pose, altitude, or rangemeter measurement.

The work is divided into two parts. The first part focuses on the dataset and the second one focuses on the spiking neural networks and egomotion estimation. To illustrate the process, Figure 1.4 is included, which also contains the relevant chapters. The literature study can be found in the Appendix. Last Appendix E is left for additional results, which could not be incorporated into the main body of the thesis due to time constraints.

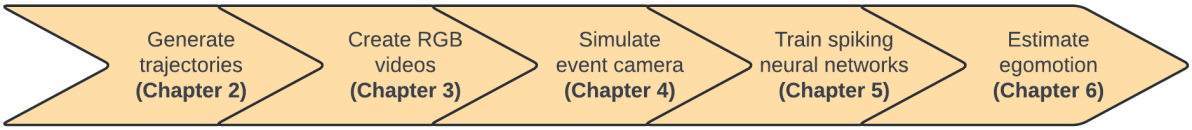



Figure 1.4: Main steps and the relevant chapters

# Part I

# Event-based landing dataset

## Overview

This part of the report will introduce the event-based landing dataset together with its generation and validation processes. Figure 1.5 provides us with a rough outline of this task.

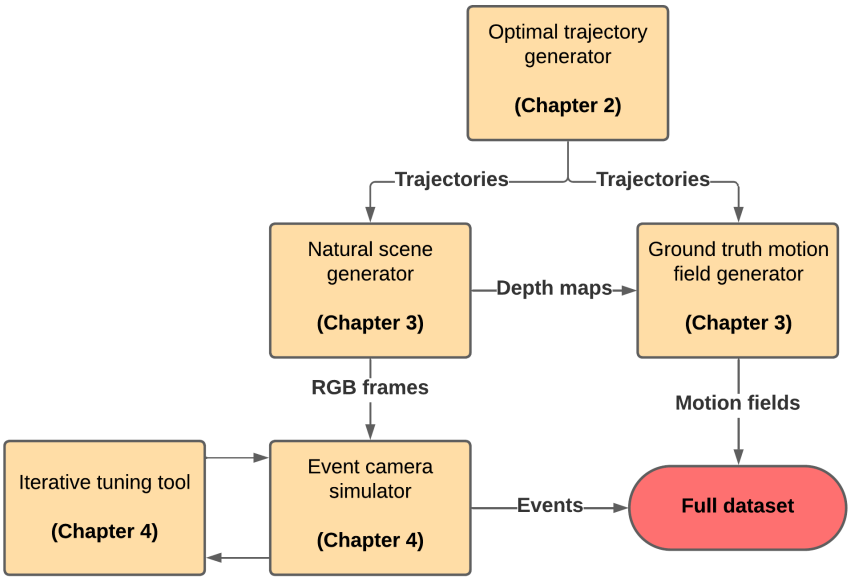



Figure 1.5: Overview of the event-based landing dataset generation and validation process

Chapter 2 will guide the reader through the generation of optimal trajectories. First, it will introduce the coordinate system used in the entire work and the optimal landing problem being solved. Secondly, it will discuss the lander model based on Apollo missions, which is being utilized to obtain the trajectories. Then it will compare modern trajectory optimization methods to make an argument for the one chosen in this work. Afterwards, it will discuss the implementation of the optimal problem and the types of trajectories chosen for the dataset. We will conclude with a brief overview of the properties of the final trajectories themselves.

Chapter 3 will show the logic behind the creation of the lunar natural scenery for the optical navigation system and ground truth motion fields. It will start with introducing a pinhole camera model and proceed with a description of the landing site and the lighting conditions. Then it will focus on the use of Planet and Asteroid Natural scene Generation Utility (PANGU, Martin et al. (2019)) to render the lunar scenery and depth maps, required for the ground truth motion fields. In the end, the motion fields themselves are discussed.

Lastly, Chapter 4 will raise the topic of event camera simulations and validation and tuning of interaction between an event camera pixel and the surface. First, it will bring about a short trade-off between event camera simulators, which will justify the choice. Secondly, the model of the event camera non-idealities used in this work will be presented. Third, validation metrics and available ground truth will be introduced for the event camera pixel-surface interaction model. Lastly, the validation and tuning process will be described.

# Chapter 2

# Optimal trajectories

This chapter will focus on the topic of generating optimal trajectories for the dataset. First, the coordinate system will be introduced together with the optimal control problem in Section 2.1. Secondly, the Apollo lander will be elaborated upon and numerical values for the problem will be presented in Section 2.2. In addition, an explanation for the choice of the optimization algorithm is shown in Section 2.3 and an overview of its implementation will be provided in Section 2.4. Then, three distinct trajectory types used in this work will be shown in Section 2.5 and the final resultant trajectories presented in Section 2.6.

## 2.1 Coordinate system and the optimal landing problem

This section will first introduce a coordinate system that will be used throughout the entire work and then a formal introduction of the optimal control problem will follow. As the dataset generation pipeline used in this work loosely follows on the bases of Azzalini et al. (2023), the reference frame chosen is taken from this source together with most of the notation in this area.

### 2.1.1 Coordinate system

An inertial reference frame $\mathcal{F}_i$ $(O_i, x, y, z)$ is placed with its origin at the landing point on the lunar surface. A non-inertial reference frame $\mathcal{F}_b$ $(O_b, X, Y, Z)$ is chosen as the lander body frame. The rotation transformation between frames can be described by the sequence of Euler angles $\phi \rightarrow \theta \rightarrow \psi$. The lander position and velocity in $\mathcal{F}_i$ are, respectively, noted as follows

$$\begin{aligned} \mathbf{r} &= [x, y, z]^T \\ \mathbf{v} &= [v_x, v_y, v_z]^T \end{aligned} \tag{2.1}$$

The reference frames are illustrated in Figure 2.1.

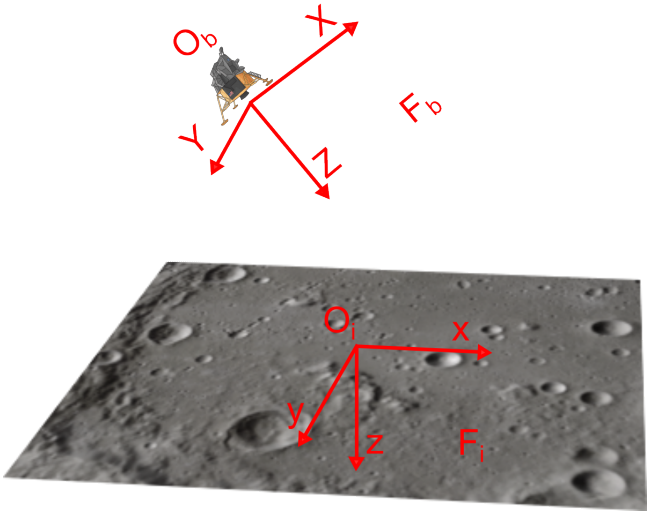



Figure 2.1: The two reference frames used in the work, inspired by Azzalini et al. (2023)

### 2.1.2 Optimal landing problem

A lander should be considered starting in a position $\mathbf{r}(0)$ with a defined initial velocity $\mathbf{v}(0)$ and angular velocity $\boldsymbol{\omega}(0)$. The lander has initial mass $m(0)$ and can vary the throttle $u_t(t) \in [0,1]$ to exert thrust $\bar{F}_a u_t(t)$ with exhaust velocity $k_e$. The angular motion of the vehicle can be controlled by six thrusters at distance $L$ from the center of gravity with throttle levels $u_{\phi,p}, u_{\phi,n}, u_{\theta,p}, u_{\theta,n}, u_{\psi,p}, u_{\psi,n}$ which have a maximum thrust magnitude of $\bar{F}_b$. The vehicle has moment of inertia $\mathbf{I}$. A trajectory is to be found from this initial state to a final state where $\mathbf{r}(t_f) = 0$ and $\mathbf{v}(t_f) = 0$ while minimizing an objective function $J = m(0) - m(t_f)$ with time $t_f$ being free.

Transformation from the reference frame $\mathcal{F}_b$ to the reference frame $\mathcal{F}_i$ can be carried out as shown below.

$$\mathbf{r} = \mathbf{R}(\phi, \theta, \psi)\mathbf{r_b}, \tag{2.2}$$

$$\mathbf{v} = \mathbf{R}(\phi, \theta, \psi)\mathbf{v_b}, \tag{2.3}$$

$$\boldsymbol{\omega} = \mathbf{R}(\phi, \theta, \psi)\boldsymbol{\omega_b} = \mathbf{R}(\phi, \theta, \psi) \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{2.4}$$

The matrix $\mathbf{R}(\phi, \theta, \psi)$ is expressed as follows.

$$\mathbf{R}(\phi, \theta, \psi) = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{2.5}$$

The lander dynamics can then be described using the following set of equations, directly adapted from Azzalini et al. (2023). The change in relation to the original work lies in treating the thrusters as six independent units, which is beneficial with regards to optimization using common optimizers which first explore the boundary conditions.

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{3}$$

$$\dot{\mathbf{v}} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - \frac{1}{m}\mathbf{R} \begin{bmatrix} 0 \\ 0 \\ \bar{F}_a u_t \end{bmatrix}, \tag{4}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \tag{5}$$

$$\mathbf{I}\dot{\boldsymbol{\omega}} = \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} + 2L\bar{F}_b \begin{bmatrix} u_{\phi,p} - u_{\phi,n} \\ u_{\theta,p} - u_{\theta,n} \\ u_{\psi,p} - u_{\psi,n} \end{bmatrix}, \tag{6}$$

$$\dot{m} = -\frac{\bar{F}_a u_t + 2F_b(u_{\phi,p} + u_{\phi,n} + u_{\theta,p} + u_{\theta,n} + u_{\psi,p} + u_{\psi,n})}{k_e}. \tag{2.6}$$

In terms of constraints, it is advisable to limit maximum angular deviation between the direction of main engine thrust and the z-axis of reference frame $\mathcal{F}_i$ in order to ensure that the vehicle does not point into free space, which is useless for optical navigation. The formulation of this constraint can be derived from Malyuta, Reynolds, et al. (2021) and can be found in the code of Azzalini et al. (2023) as follows, with $\gamma$ being the maximum tilt angle which was chosen to be 1 radian for the purpose of this work.

$$cos(\phi)\cos(\theta) >= cos(\gamma) \tag{2.7}$$

Moreover, for the problem to be physically realistic it must hold that the spacecraft is above the ground for the entire trajectory.

$$z < 0 \tag{2.8}$$

For purposes of numerical convergence with some optimization algorithms, it was found to be beneficial to also give upper and lower bounds to the pitch, yaw and roll angles. However, these values were set so high as to never be realistically achieved.

## 2.2 Apollo lander model and its trajectories

The Apollo Lunar Module was chosen as a vehicle for this work because in the publicly accessible domain, it is the best documented lunar lander. As such, it is possible to reconstruct the model described in Subsection 2.1.2 with high fidelity and, despite being obsolete, it is still considered better to use the Apollo than merely guess how such values should look for a more modern spacecraft. This section will describe the reconstruction of physical characteristics of this lander.

### 2.2.1 Mass and inertia

We consider the vehicle at the high gate during the landing trajectory, so there is an uncertainty about the exact mass and moment of inertia figures as they depend on the amount of fuel consumed. For simplicity an assumption is adopted, similar to the one used in code of Azzalini et al. (2023) that all these values are close to 60 percent of what they were at the start of the descent. Stengel (1969) lists the initial mass at 15000 kg and the initial values of pitch, roll and yaw moments of inertia to be 34000 $kgm^2$, 33900 $kgm^2$ and 31200 $kgm^2$ respectively. These values scaled down to 60 percent are listed in Table 2.1 and used in the simulations.

| **Parameter** | **Symbol** | **Value** | **Source** |
|---|---|---|---|
| Mass | $m$ | 9000 kg | Stengel (1969) |
| Yaw moment of inertia | $I_{xx}$ | 20340 $kgm^2$ | Stengel (1969) |
| Pitch moment of inertia | $I_{yy}$ | 20400 $kgm^2$ | Stengel (1969) |
| Roll moment of inertia | $I_{zz}$ | 18720 $kgm^2$ | Stengel (1969) |

Table 2.1: Apollo mass and inertia properties used in the simulations.

### 2.2.2 Main engine and thrusters

Bartlett et al. (1966) lists the maximum thrust of the Apollo Lunar Module descent engine at 46.7 kN. The minimum thrust of 4.7 kN is not modeled. The specific impulse of the engine is mentioned to be 306 s by Avvenire (1974) from which exhaust velocity equal to 3002 m/s can be easily calculated.

With regard to thrusters, Stengel (1969) notes that the Reaction Control System (RCS) on the Apollo Lunar Module consisted of two eight-thruster systems with each of the thrusters having a maximum thrust of 445 N at fuel consumption of 0.16 kg/s which leads to exhaust velocity of 2782 m/s. The exhaust velocity value is considered close enough to the one for main engine to not be differentiated. Both systems could be fired at once if necessary. Moreover, Stengel (1969) claims the resultant moment from firing each roll and pitch thrusters to be equal to 746 Nm, while for the yaw jets this value is equal to 695 Nm, leading to respective distances of 3.35 m and 3.12 m between the thruster. For simplicity, the value of 3.35 m is considered for all thrusters. Table 2.2 summarizes all these parameters.

| **Parameter** | **Symbol** | **Value** | **Source** |
|---|---|---|---|
| Main engine max thrust | $\bar{F}_a$ | 46.7 kN | Bartlett et al. (1966) |
| RCS max thrust | $\bar{F}_b$ | 445 N | Stengel (1969) |
| Exhaust velocity | $k_e$ | 3002 m/s | Avvenire (1974) |
| Distance between RCS thrusters | $L$ | 3.35 m | Stengel (1969) |

Table 2.2: Apollo engine and RCS parameters used in the simulations.

### 2.2.3 Apollo trajectories

For the purposes of the work, only the part of the trajectory between the so-called high gate and low-gate is considered. As Bennett (1970) notes, high gate is the end of the approach phase and point where the final landing maneuver is initiated. On the other hand, low gate is a point very close to the ground, where close range methods of navigation tend to be used.

For the Apollo mission, Bennett (1970) lists the high gate altitude at 2286 m (7500 feet) and the low gate altitude at 152 m (500 feet). Furthermore, Bennett (1970) notes that at the high gate, the Apollo Lunar Lander closed to the ground with a vertical velocity of 44 m/s (145 fps) and a total velocity of 156 m/s (506 fps) which suggests a horizontal velocity component of 148 m/s (484 fps). These parameters were further processed to serve as an inspiration for the initial trajectory parameters listed in Table 2.4.

## 2.3 Choice of the optimization method

The complicated problem described in Subsection 2.1.2 can be only solved using non-linear programming further elaborated upon in Appendix B. This is a cumbersome process and a more elegant formulation of the problem with simpler solution would be optimal. A possible opportunity lies with the family of convex optimization methods. This section presents results of investigation into the use of the famous member of this family, the G-FOLD algorithm by Açıkmeşe et al. (2012), for this purpose.

### 2.3.1 Description of G-FOLD

As outlined in Appendix B, if a lander is considered to be a lumped mass and all state and control constraints are one-sided inequalities, convex optimization can be leveraged for the problem. This brings several advantages, such as guaranteed convergence, where solution is feasible and much lower computational complexity. One of the well-used algorithms in this area is G-FOLD, which also utilizes so-called lossless convexification to allow for formerly non-convex bounds on thrust, pointing constraints, and glideslope constraints.

Given a time-of-flight, G-FOLD calculates the minimum fuel trajectory. As the optimal flight time is unknown, it is necessary to perform line search, which is, however, quite unproblematic, as the algorithm itself is rather inexpensive. If it is assumed that the nozzle is not gimballing, but is fixed to the spacecraft, then the thrust direction can be used as a proxy to recover the attitude information from the algorithm.

### 2.3.2 Comparison of G-FOLD and general non-linear programming

To compare G-FOLD with general non-linear programming, an example test case was generated with the Apollo model described in Section 2.2 and initial conditions summarized in Table 2.3.2. The full problem from Subsection 2.1.2 is being solved by the non-linear program, while G-FOLD is to solve its variant without any attitude dynamics involved.

| **Parameter** | **Value** | **Unit** |
|---|---|---|
| $x(0)$, $y(0)$, $z(0)$ | 0, 0, 2300 | m, m, m |
| $v_x(0)$, $v_y(0)$, $v_z(0)$ | 70, 0, 44 | m/s, m/s, m/s |
| $\psi(0)$, $\theta(0)$, $\phi(0)$ | 0, 0, 0 | rad, rad, rad |
| $z(t_f)$, $\psi(t_f)$, $\theta(t_f)$, $\phi(t_f)$ | 0, 0, 0, 0 | m, rad, rad, rad |

Table 2.3: Initial and final conditions for the test case to compare G-FOLD with a non-linear program

The non-linear program is based on the code of Azzalini et al. (2023) and implemented using the AMPL parser (Fourer (1996)) and the SNOPT solver (Gill et al. (2005)) and it is further elaborated in Subsection 2.4.1. On the other hand, G-FOLD is implemented using the CVXPY parser (Diamond and Boyd (2016)) and the ECOS solver (Domahidi et al. (2013)).

After running the experiment, trajectory shapes and velocity profiles were found to be extremely similar. There is never a distance exceeding more than 100 meters between the two simulated vehicles and the average discrepancy values are one order of magnitude lower than that. However, the same cannot be said about the attitude of the vehicle. To truly test how precisely the G-FOLD attitude proxy in the form of the thrust vector copies the real attitude dynamics of the vehicle, a somewhat unnatural starting position was chosen where the vehicle is completely upright in the beginning of the trajectory as can be seen from Table 2.3.2. Figure 2.2 shows the pitch angle values throughout the trajectory, and interesting conclusions can be drawn.

In Figure 2.2 it can be seen that indeed the initial and final conditions will create a mismatch between the estimated attitudes because G-FOLD will assume that thrust can be exerted in the ideal direction from the beginning and to the very last moment. On the other hand, when the attitude dynamics are simulated, it takes time for the vehicle to rotate to the optimal position for braking and to satisfy the final attitude conditions. Moreover, G-FOLD, which is not penalized for the attitude maneuvers, predicts a smooth change in the pitch angle, while the solution to the general non-linear program predicts a pitch angle which is essentially constant over a large part of the trajectory as attitude maneuvers do have some cost.

In case of navigation using optical flow, the incomplete attitude dynamics pose significant challenges. As Equation 6.1 suggests, especially at higher altitudes the resultant motion field quickly becomes dominated by

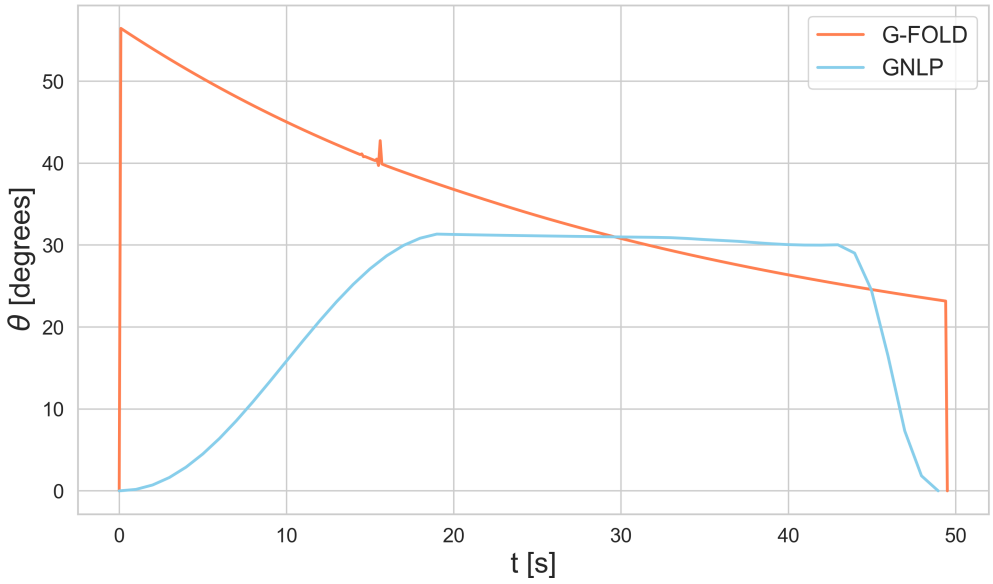



Figure 2.2: Comparison between pitch angle profile generated by G-FOLD and general non-linear programming

the rotational terms. Hence, G-FOLD is deemed inadequate for this purpose and the work proceeded with general non-linear programming.

## 2.4 Implementation of the optimization method

This section will quickly introduce the discretization of the optimal landing problem and the software applied to solve its discrete form. It will also make a note about the strategy used to obtain realistic initial conditions in all scenarios.

### 2.4.1 Problem transcription and solver

The continuous problem described in Subsection 2.1.2 is transcribed into a non-linear programming problem of a discrete fashion using Hermite-Simpson collocation like in the work of Azzalini et al. (2023). The flight is broken up into a number of temporal intervals. As stated in Moreno-Martín et al. (2024), the state trajectories are approximated by cubic polynomials in each of the intervals. To see the details of the discretization method, the reader is recommended to refer to Moreno-Martín et al. (2024),

As already outlined in Subsection 2.3.2, this discrete form of the problem can be directly handled by the AMPL parser and solved using the SNOPT interior point method optimizer. It is noted that the freeware CasADi (Andersson et al. (2019)) parser was also experimented with, but it proved inferior to AMPL in terms of reduction of redundancies in the problem. Similarly, freeware solver IPOPT (Wächter and Biegler (2006)) was tried for this purpose, but only poor convergence was achieved on this optimization problem with many dimensions.

### 2.4.2 Optimal control problem initialization

Initializing the optimal control problem is problematic. If the initial attitude needs to be aligned with the flight path in order to prevent unrealistic attitude changes, then the flight path must be known. However, that is impossible to do without knowing the flight path, creating a chicken-and-egg problem. It is possible to solve this problem by simply cutting off part of the trajectory with the initial attitude alignment, but as the number of intervals is limited by convergence problems, it leads to a decrease in the trajectory resolution.

Hence, a scheme was adapted in this work, where a “major” trajectory, which starts a few kilometers from the desired starting point, is first calculated to provide accurate initial conditions for the “minor” trajectory, which is the final product. The minor trajectory starts where the major trajectory crosses the high gate altitude $z_{hg}$. As it will be seen in Subsection 2.5.3, this arrangement also allows to simulate divert maneuvers better, where the landing point suddenly changes to a new point in the landing ellipse.

## 2.5 Trajectory types

To ensure that the trajectories provide enough variation for learning and bring insight into effectiveness of event camera in different flight regimes, three different trajectory types are considered and they are named ventral, nominal and divert type. This section will provide the respective rationale behind each of them together with a detailed description.

### 2.5.1 Ventral trajectories

The simplest possible trajectory is a purely ventral descent, where there is no horizontal velocity or attitude maneuvers present. It is notable that in such a case, optimal control is always of a bang-bang type as outlined by Meditch (1964) and the vehicle will be let to free-fall until a point where switching to maximum thrust will result in zero velocity at the surface. An unfortunate consequence of this result is that all the trajectories will exhibit the same velocity at a certain altitude $z$, if they already have the engine switched on. This may lead to potential overfitting as the last parts of the ventral trajectories exhibit identical optical flow. This behavior can be seen well in Figure 2.3 in Section 2.6.

While the ventral trajectory is not a realistic flight path for a vehicle all the way from the high gate to the low gate, it is still interesting to investigate it in order to estimate performance. Moreover, the last few hundred meters of a lunar landing trajectory may be well resembling a ventral landing scenario as e.g. shown in the work of S. Li et al. (2015).

Initial altitude $z(0)$ is sampled from $z \sim \text{Uniform}(z_{high}, z_{low})$ and vertical velocity is sampled from $v_z \sim \text{Uniform}(v_{z,low}, v_{z,high})$. High gate altitude is sampled from $z_{hg} \sim \text{Uniform}(z_{hg,high}, z_{hg,low})$

### 2.5.2 Nominal trajectories

Nominal trajectory represents a typical scenario between the high gate and the low gate, where the spacecraft reaches the high gate already aligned with the flight path to the desired landing point and proceeds as expected.

The parameters for the major trajectory are sampled in the following fashion. First, the initial ground range $R$ to the target is determined from $R \sim \text{Uniform}(R_{low}, R_{high})$ and a random azimuth angle $\kappa$ is chosen as $\kappa \sim \text{Uniform}(0, 2\pi)$. This allows to find a horizontal position of the starting point $x(0), y(0)$ as $Rcos(\kappa), Rsin(\kappa)$. Initial altitude, high gate altitude and vertical velocity are sampled identically to ventral trajectories. Horizontal velocity is distributed with $v_h \sim \text{Uniform}(v_{h,low}, v_{h,high})$. The horizontal velocity is then broken down into x and y component such that the spacecraft flies toward the origin as shown below. The spacecraft starts the major trajectory always fully vertical and with zero angular velocity.

$$v_x(0) = -v_h \cdot \frac{x(0)}{\sqrt{x(0)^2 + y(0)^2}}, \tag{2.9}$$

$$v_y(0) = -v_h \cdot \frac{y(0)}{\sqrt{x(0)^2 + y(0)^2}}. \tag{2.10}$$

### 2.5.3 Divert trajectories

Divert trajectories represent a situation where a divert needs to be conducted after passing the high gate due to e.g. the landing point being found unsuitable after closer visual inspection. These scenarios are included to make sure that there are fast attitude maneuvers and unusual trajectory shapes included in the dataset.

The diverts are generated using the following scheme. First, initial conditions for a minor trajectory are determined in a fashion identical to the process for nominal trajectory shown in Subsection 2.5.2. For the minor trajectory, the landing point is shifted to a new random point within a circle with radius $R_{div}$. This may generate significant maneuvers as the spacecraft alters its path away from the path set by the initial conditions.

## 2.6 Resultant trajectories

This section will describe the initial conditions used to generate the resultant trajectories and it will continue to present a short overview of the trajectories themselves.

### 2.6.1 Trajectory initial conditions

Based on the description of the Apollo trajectories in Section 2.2, parameters and parameter ranges for the simulations were set. They are listed in Table 2.4. It is noted that, while there was an effort to keep everything close to the Apollo trajectories, certain liberties were taken with the ranges to ensure that the trajectories are sufficiently varied.

| **Parameter** | **Symbol** | **Value/Range** |
|---|---|---|
| Ground range min | $R_{low}$ | 5 miles (8.04 km) |
| Ground range max | $R_{high}$ | 7 miles (11.26 km) |
| Initial altitude | $z_{low}, z_{high}$ | -10000 feet, -12000 feet (-3048 m, -3657 m) |
| Horizontal velocity magnitude | $v_{h,low}, v_{h,high}$ | 120 m/s, 200 m/s |
| Initial vertical velocity | $v_{z,low}, v_{z,high}$ | 20 m/s, 80 m/s |
| Maximum divert range | $R_{\text{div}}$ | 1 mile (1.609 km) |
| Minor trajectory start altitude | $z_{hg,low}, z_{hg,high}$ | -7400 feet, -7600 feet (-1950 m, -2560 m) |

Table 2.4: Parameters used to generate major trajectories and minor trajectory start altitude

### 2.6.2 Trajectory overview

Current state-of-the-art event-based datasets like Stereo Event Camera Dataset for Driving Scenarios (DSEC, M. Gehrig, Aarents, et al. (2021)) often contain at least half an hour of labeled camera recordings, which is demonstrably enough for learning in smaller networks. Since the flight time from the given initial conditions to the ground will typically be around one minute, it was decided to generate 36 trajectories with 12 of them belonging to each type.

Figure 2.3 shows the variation of vertical and horizontal velocity with altitude in different trajectories. This, as can be seen from Equation 3.4, is the most important parameter as it will lead to a varied optical flow and visual impression from the respective trajectories. It can be observed that a wide array of options is covered and especially the divert trajectories add very unusual velocity profiles, which should reduce chances of overfitting and allow to establish reliability of inference in different flight regimes.

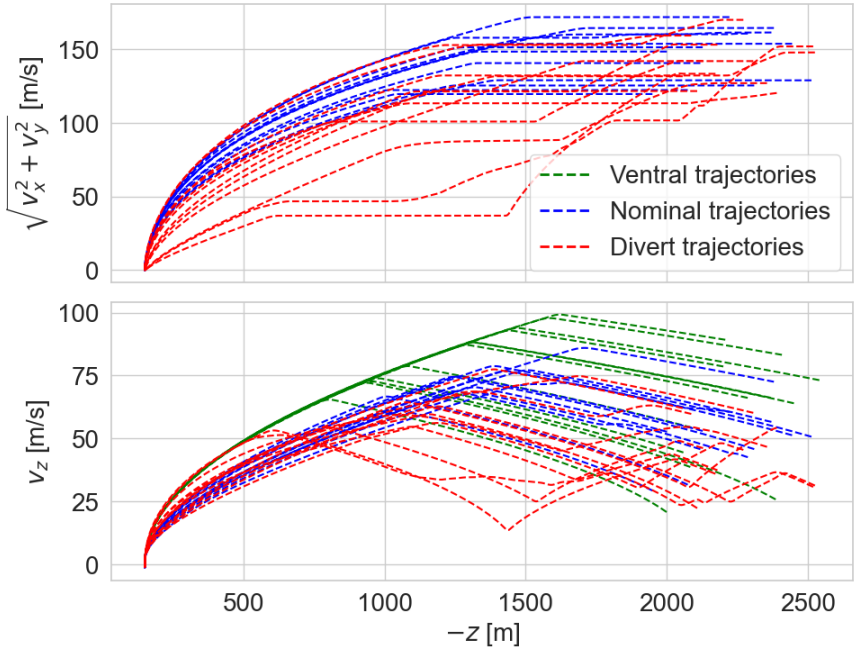



Figure 2.3: Vertical and horizontal velocities on the dataset trajectories plotted against altitude.

Figure 2.4 provides an overview of angular velocities against altitude on different nominal and divert trajectories. What is noticeable are significant attitude maneuvers throughout large part of all the divert trajectories and very large angular velocities at the low gate for all of the trajectories.

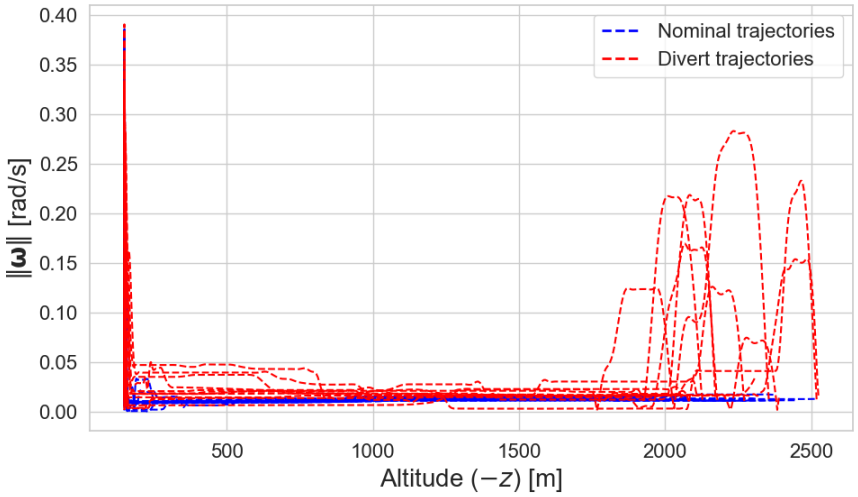



Figure 2.4: Angular velocities on the dataset trajectories plotted against altitude.

# Chapter 3

# Surface model and ground truth motion fields

This chapter will introduce the visual model of the surface used for the investigations and the event camera simulator. First, it will elaborate upon the pinhole camera in Section 3.1 and the choice of the landing site and the surrounding environment in Section 3.2. Secondly, it will investigate how the surface can be generated using the Planet and Asteroid Natural Scene Generation Utility (PANGU) utility by Martin et al. (2019) in Section 3.3. Lastly, the ground truth motion fields will be discussed in Section 3.4.

## 3.1 Pinhole camera

For simplicity, this work considers all cameras to act as ideal pinhole cameras, thus free of any distortions. This means that point $X, Y, Z$ in the $\mathcal{F}_b$ frame (please refer to Subsection 2.1.1 for the specification of the reference frames) will project into the point $\hat{x}, \hat{y}$ on the image plane according to Equation 3.1 with $f$ being the focal length of the camera.

$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = -\frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{3.1}$$

The projection of a point on an image plane is further illustrated by Figure 3.1. Equation 3.1 can be easily derived from this figure using similar triangles.

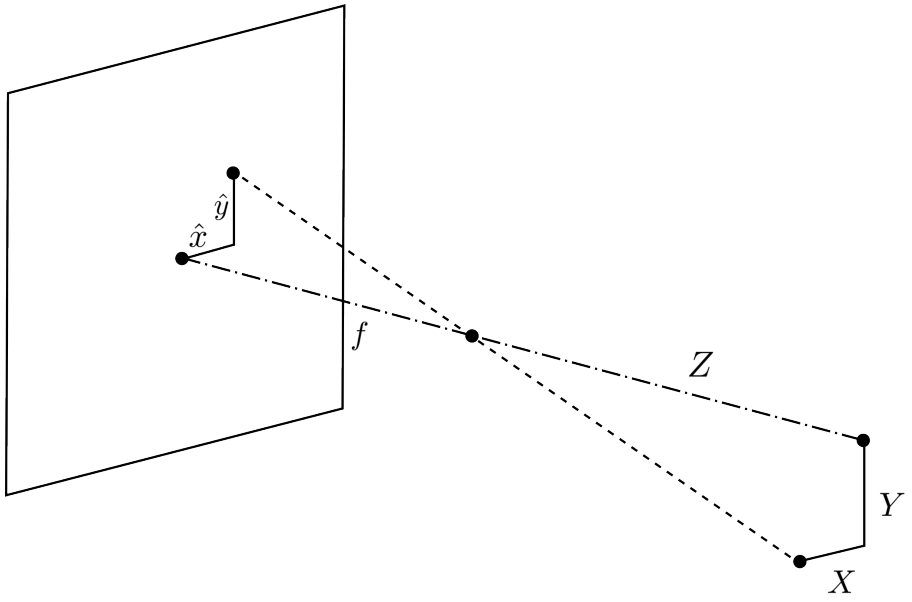



Figure 3.1: Projection of a point in the non-inertial frame $\mathcal{F}_b$ on the image plane

An important parameter is the field of view of the camera and its resolution. In this work, all cameras are considered to have a field of view $FOV$ of 45 degrees and a resolution $W$x$W$ of 200 x 200 pixels. As Azzalini

et al. (2023) shows, the focal length of a pinhole camera can be expressed from these parameters as outlined below.

$$f = \frac{W}{2tan\left(\frac{FOV}{2}\right)} \tag{3.2}$$

## 3.2 Landing site and lighting conditions

This section will first briefly introduce the chosen landing site and motivate this choice. Then it will proceed to describe the lighting conditions used in this work.

### 3.2.1 Landing site

The region chosen for the landing site in the simulated trajectories is the area around the lunar south pole. This part of the Moon is particularly interesting from the point of view of optical navigation. There is a very interesting interaction of areas of extreme illumination and darkness in the craters, as pointed out in Märtens et al. (2024). Moreover, it is also a region of great scientific interest due to the believed presence of water ice as stated in Wei et al. (2023). As a result, it is both the target of current landing missions like the Nova-C lander described in Foust (2024) and future missions like the Chang'e 7 lander, which is outlined in Wei et al. (2023).

The specific landing site chosen for the simulation is the Sverdrup crater, which lies essentially at the lunar south pole. It was chosen due to the immediate availability of a high-quality model by Martin et al. (2019), its varied surroundings and its very flat terrain shown in Figure 3.2, which makes Sverdrup a good landing site.

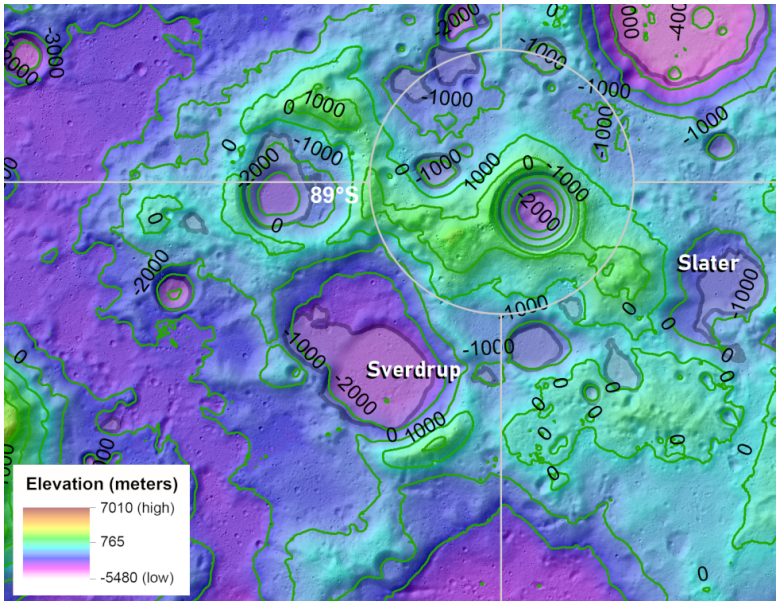



Figure 3.2: Topographic map of the Sverdrup crater showing its flat floor and varied surroundings. Image taken and modified from Stopar and Meyer (2019).

### 3.2.2 Lighting conditions

In reality, Sverdrup crater is covered in perpetual darkness as pointed out in Jia et al. (2022), but for the purpose of the simulations, the solar elevation angle was increased to 10 degrees, such that the terrain is suitable for optical navigation.

The 10 degrees angle was chosen to keep a rough similarity to the ground truth data used for the validation coming from work of Märtens et al. (2024). Use of this data is further elaborated in Section 4.3. It is noted that with such solar elevation, nearly all craters still contain areas of extreme darkness.

## 3.3 Rendering of lunar scenery

This section will focus on the use of PANGU to generate RGB imagery, which represents what would be seen by a camera on a lunar lander during the landing maneuver. First, the surface model and its shortcomings will be discussed. Afterward, a short discussion is added about challenges of interpolating the image frames in order to generate large-scale datasets with high frame rates efficiently.

### 3.3.1 Surface model

As already stated, the surface representation is generated using PANGU. The PANGU software can generate a photorealistic scene using a Digital Elevation Model (DEM). In case of this work, a DEM with 240 m/px resolution was used, which was based on the data from the Lunar Reconnaissance Orbiter. As such a resolution is clearly too coarse for the purposes of optical navigation within kilometers or hundreds of meters from the Moon, further enhancements were used in form of additional craters and boulders.

A significant disadvantage of PANGU is that it uses the Discrete Level of Detail method to increase the efficiency of rendering. This leads to a highly undesirable popping effect, where the level of detail suddenly increases within a larger area between two frames. The difference between the low and the high level of detail regions is demonstrated in Figure 3.3. This issue is however, believed to not have significant adverse effects on the work, because the inspection has revealed that circa only one out of 200 to 300 frames is affected by the sudden popping phenomenon if the simulations are run at 100 fps frame rate.

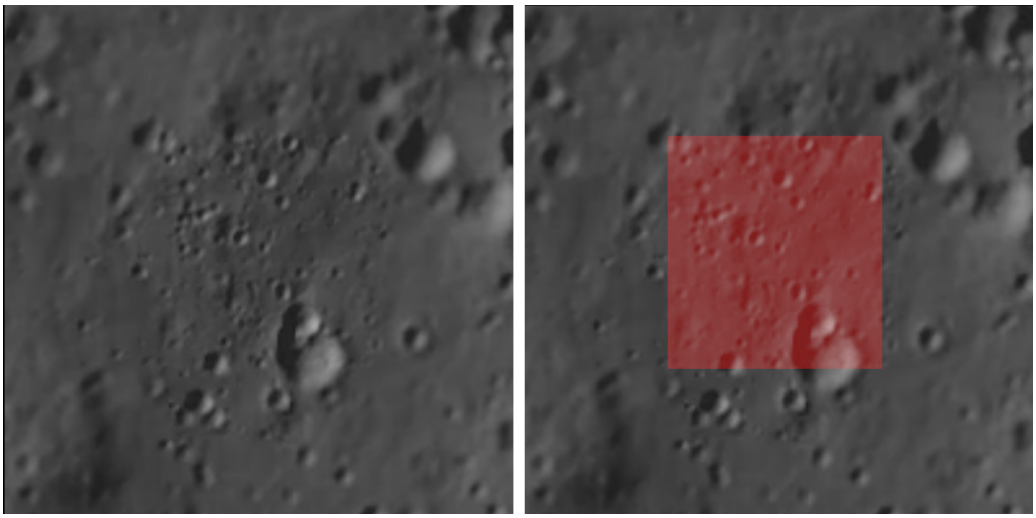


Figure 3.3: PANGU rendered regions with a different level of detail, the area marked red on the right is full of craters and small scale features while the rest of the image is much more coarse.

### 3.3.2 Frame interpolation

It is noted that the surface rendering can be extremely consuming in terms of computational time and finishing rendering of tens of final landing trajectories at frame rates appropriate for event camera simulators, which often exceed 100 fps, can easily take days on a common GPU, if appropriate measures are not taken. In order to mitigate this issue, experiments were made with motion-aware sampling and frame interpolation to decrease the rendering efforts.

For the motion-aware sampling, the method proposed by Rebecq et al. (2018) is used. The timestamp of the $(k+1)$-st sample is calculated as follows, with $\lambda$ being a tuning constant and $\sqrt{u_{\hat{x}}^2 + u_{\hat{y}}^2}_{max}$ being the maximum motion field magnitude at the time of sample $k$. Motion field can be calculated according to Equation 3.4. As such, the timestamps are very dense when there is significant observable movement, and vice versa.

$$t_{k+1} = t_k + \frac{\lambda}{\sqrt{u_{\hat{x}}^2 + u_{\hat{y}}^2}_{max}} \tag{3.3}$$

Three readily available interpolation methods were evaluated in order to reproduce high frame-rate imagery from the samples.

- Option 1: Interpolation using the SuperSloMo neural network by Jiang et al. (2018)
- Option 2: Interpolation using a block matching algorithm implemented in ffmpeg by Tomar (2006).
- Option 3: Creating the intermediate frames by warping according to the known motion field using the oflibpytorch library of Ravasio et al. (2021)

Options 1 and 2 do not make use of the known motion field. That makes them much more prone to error and leads to undesired artifacts, such as "wavy" edges of the craters. Option 3, interpolation by warping the original frames according to the known motion field, results in a much better estimate and should be preferred. Yet, it introduces steps in intensity visible on an event camera, whenever an original frame, which is not a product of interpolation, is passed. This is due to the forward or backward nature of the simple warping schemes, which always take only one original frame into account.

Due to these issues a decision was made to rather not interpolate the frames and simply generate the surface imagery at the lowest acceptable frame rate, which was deemed to be 100 fps based on the simulator characteristics described in Section 4.1 and input nature to the processing methods described by Subsection 5.3.1. Moreover, as already mentioned, the relatively modest resolution of 200x200 pixels was chosen to speed up the rendering.

## 3.4 Ground truth motion fields

This section will focus on the topic of ground truth motion fields. First, it will show how the ground truth motion fields can be calculated using trajectory information and the depth map. After, it will assess how depth maps, necessary for generating such motion fields, can be generated using PANGU LIDAR simulation tool.

### 3.4.1 Estimating motion fields

As Grabe et al. (2015) notes, the motion field in point $\hat{u}(\hat{x}, \hat{y})$ can be expressed using Equation 3.4 with $Z$ being the depth of a scene point, which projects into point $\hat{x}$, $\hat{y}$ in the image plane.

$$\hat{\mathbf{u}}(\hat{x}, \hat{y}) = \mathbf{K}(\hat{x}, \hat{y})\boldsymbol{\omega} + \frac{1}{Z}\mathbf{L}(\hat{x}, \hat{y})\mathbf{v_b} \tag{3.4}$$

The rotation-related matrix $\mathbf{K}(\hat{x}, \hat{y})$ and translation-related matrix $\mathbf{L}(\hat{x}, \hat{y})$ in Equation 3.4 can be expressed as follows according to Grabe et al. (2015).

$$\mathbf{K}(\hat{x}, \hat{y}) = \begin{bmatrix} -\hat{x}\hat{y}/f & (f^2\hat{x} + \hat{x}^2)/f & -\hat{y} \\ -(f^2\hat{y} - \hat{y}^2)/f & \hat{x}\hat{y}/f & \hat{x} \end{bmatrix} \tag{3.5}$$

$$\mathbf{L}(\hat{x}, \hat{y}) = \begin{bmatrix} f & 0 & -\hat{x} \\ 0 & f & -\hat{y} \end{bmatrix} \tag{3.6}$$

### 3.4.2 Generating depth maps

As it can be seen from Equation 3.4, it is necessary to know the accurate $Z$ coordinates of all the points in the field of view to obtain an accurate ground truth motion field. For this purpose, the PANGU feature called "snapshot LIDAR" can be utilized. The "snapshot LIDAR" provides the slanted range $l_s(\hat{x}, \hat{y})$ to every surface point in the field of view with a reasonably low rendering time similar to the one of a normal RGB image. It achieves this performance by using a rasterization-based algorithm which, albeit less precise than the ray-tracing options commonly found for LIDAR simulation, can still yield sufficient precision for the purpose at hand where sub-meter errors have almost no influence.

From the pinhole camera geometry, shown in the Figure 3.1 it can be easily seen that Equation 3.7 is valid.

$$Z(\hat{x}, \hat{y}) = \frac{f l_s(\hat{x}, \hat{y})}{\sqrt{\hat{x}^2 + \hat{y}^2 + f^2}} \tag{3.7}$$

# Chapter 4

# Event camera simulation and model validation

This chapter will first justify the choice of an event camera simulator and present test results. Afterwards, the event camera model used in this work will be described. Then it will follow to establish limitations of this model. In the second half, it will concern itself with the validation procedure. First, available ground truth data will be discussed. The validation metrics will be established. Lastly, the process itself and its results will be introduced.

## 4.1 Event camera simulator testing

Based on the discussion in Appendix A, five simulators were selected for the testing as they represent the state-of-the-art in different approaches to event camera simulation. They are summarized in the Table 4.1.

| **Simulator** | **Source** |
|---|---|
| v2e | Hu et al. (2020) |
| DVS Voltmeter | Lin et al. (2022) |
| V2CE | Z. Zhang et al. (2023) |
| IEBCS | Joubert et al. (2021) |
| EventGAN | Zhu et al. (2019) |

Table 4.1: Event camera simulators chosen for testing

### 4.1.1 Test method

The chosen test case is a calibration file from dataset of Mueggler et al. (2017) featuring an event camera moving around a chessboard in a natural light indoor scene. It was selected because it contains sharp movement and a lot of contrasting features which makes it easy to simulate effects. The sensor used to capture the ground truth is the DAVIS 240 event camera with an active pixel frame. First second of the recordings was used to fit the simulator hyper-parameters and following four seconds to evaluate the behavior. The video was artificially upsampled from 30 frames per second to 100 frames per second using Super SloMo Jiang et al. (2018).

Fitting the hyper-parameters is fairly non-trivial. However to assess the qualities of the simulators, it must be done to a similar level for all the subjects of examination. As Lin et al. (2022) suggests, there is no clear metric to compare two point clouds and both the event count and timing of the events matter. Therefore the following method was used. The aforementioned one second long sample from the ground truth was fed together with a one second long simulated sample into objective function $L$ described by the Equation 4.1. Let $\mathcal{E}_{sim}$ be the set of simulated events and $\mathcal{E}_{gt}$ the set of ground truth events. The selected objective function with tuning weights $C_1$ and $C_2$ weighs a timing term represented by Wasserstein 1-distance between distributions of time intervals $\tau_{sim}$ and $\tau_{gt}$ between events on individual pixels (a timing metric proposed by Lin et al. (2022)) and the sum of absolute differences between total number of positive and negative events generated in the one second time-frame. This objective function was then optimized using Bayesian Adaptive Direct Search method (Singh and Acerbi (2024)) which is very well suited for highly non-linear black box optimization problems.

The author notes that the results of the optimization could definitely be improved, however this method is considered sufficient to reveal most of the inherent qualities and deficiencies which the simulators possess.

$$L = C_1 W_1(\tau_{sim}, \tau_{gt}) + C_2(|n(e \in \mathcal{E}_{sim}, p = 1) - n(e \in \mathcal{E}_{gt}, p = 1)| + |n(e \in \mathcal{E}_{sim}, p = -1) - n(e \in \mathcal{E}_{gt}, p = -1)|) \tag{4.1}$$

A special case is V2CE which does not have any parameters to tune as it fully relies on a pre-trained neural network. It is therefore used with a model based on DAVIS 346 camera which is created by the simulator authors and trained on the MVSEC dataset which includes mostly motion on vehicles in natural light (Zhu, Thakur, et al. (2018)).

The metrics being investigated in the testing are the following:

- Computational cost
- Timestamp fidelity
- Event count and distribution fidelity
- Ease-of-use and flexibility

While computational cost is fairly straightforward, other metrics require some explanation. There is a good reason to separate fidelity of event distribution and the fidelity of timestamps. As it can be seen in Appendix D, large part of the relevant algorithms for event processing essentially discards the precise timing information and utilizes only the event count-per-pixel over certain time interval. On the other hand some model-based algorithms are very sensitive to precise timing information. Therefore it can be said that the importance of the timestamp fidelity differs based on what is the output used for. Moreover there can be some sort of systematic errors like e.g. events clustered around discrete timestamps which can have potentially an impact on learning algorithms. The ease-of-use is very important in a time constrained project where main focus is on other topics. Flexibility is also of major importance as the model needs to be plausibly adapted to lunar landing conditions.

### 4.1.2 Computational cost

The tests were carried out on a desktop with Intel Core i7-8750H CPU and 16 GB installed RAM. While it is noted that it would be better to run the tests on a modern GPU, there were technical challenges with setting up IEBCS in such an environment. However, in theory all the simulators are ready to be run on a modern GPU. The results are summarized in Table 4.2.

| **Simulator** | **Frames processed per second** |
|---|---|
| v2e | 11 |
| DVS Voltmeter | 22 |
| V2CE | 1 |
| IEBCS | 21 |
| EventGAN | 2 |

Table 4.2: Frames processed per second for the different event camera simulators under testing.

It can be seen that v2e, DVS Voltmeter and IEBCS have little appreciable difference between each other. V2CE and EventGAN will be an order of magnitude slower than all of the other simulators. EventGAN is also excluded from further discussion as it shows very little advantage over comparable V2CE in computational costs, while having several major disadvantages over its peer such as a complete lack of precise timestamps.

### 4.1.3 Timestamp fidelity

Figure 4.1 shows samples of point clouds from the ground truth and each simulator for qualitative comparison. On first glance it can be seen that v2e generates the events around discrete timestamps which are evenly distributed between the interpolated frames. V2CE, in accordance with the expectation, creates a point cloud which fairly resembles the ground truth in its nature. IEBCS clusters the events in a manner rather similar to v2e, however it does fill the in-between spaces with many additional events. Lastly the DVS Voltmeter creates a point cloud showing a lot of resemblance to the ground truth in its form.

Comparison of time between sample consecutive events histograms shown in Figure 4.2 tells another part of the story. The v2e histogram does indeed result in a number of discrete spikes in the histogram caused by the

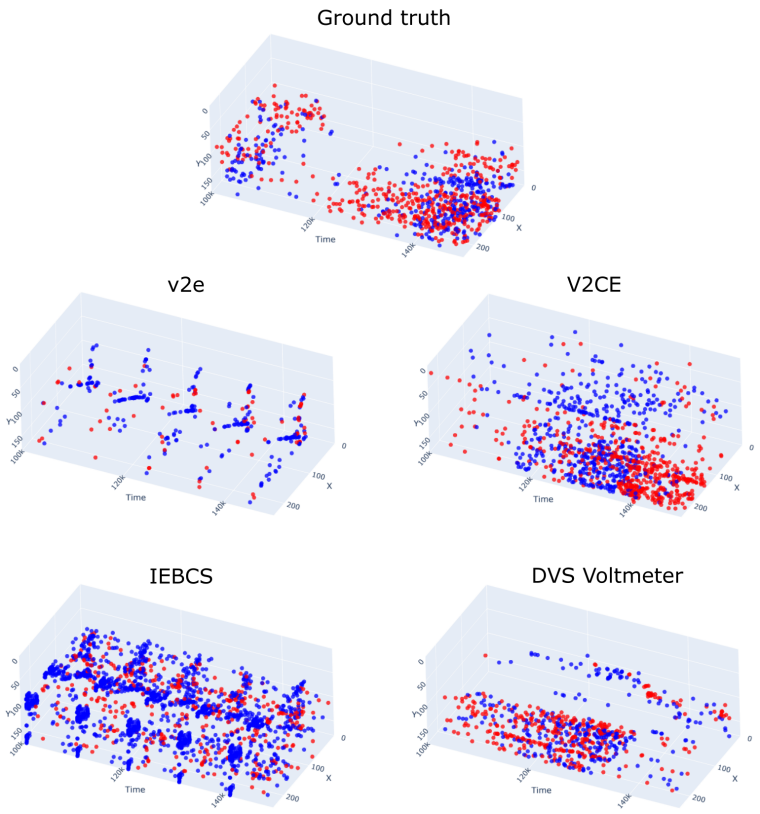



Figure 4.1: Point cloud samples generated by different event camera simulators. Of note is a tendency to cluster the events close to discrete timestamps shown by v2e and IEBCS.

tendency to cluster to discrete timestamps. On the other hand, e.g. in this case it seems that the interpolated frame rate is high enough to ensure that pixel temporal dynamics can largely be simulated well even in this discrete manner. A drawback however is the inability of v2e to generate multiple events on one pixel in a very quick succession. As can be seen from the ground truth, often a pixel fires multiple events in a very short time-span. The ability to simulate this behavior is in principle missing in v2e.

The IEBCS shows slight problems with a spiky histogram similar to v2e albeit significantly less pronounced. The event histogram is rather similar to the ground truth, but it was found extremely challenging to match the thickness of the "tail" of the distribution.

V2CE also offers very event camera-like timestamps. The shape of the histogram indeed very well corresponds to the ground truth. However it is impossible to adjust the parameters for the timing of events to fit a certain distribution without having a very substantial amount of data to retrain the underlying neural network which can result in significant disparities in some cases.

DVS Voltmeter tends to generate timestamp distributions closest to the ground truth. As will be discussed later, however, this may come at a price of significant over-fitting to certain conditions under which the event camera is used.

### 4.1.4 Event count and distribution

Surprisingly it was found that while it is no problem to fairly precisely match the number of events in the training slice of the video, it is often very difficult to extrapolate this performance to the test slices. As the numbers were found to vary a lot based on the sample, it was decided to not include a quantitative analysis here. However it is noted that all of the simulated event counts often differed in the order of low tens of percent while V2CE showed the lowest discrepancies and IEBCS the highest (sometimes as much as 60 percent).

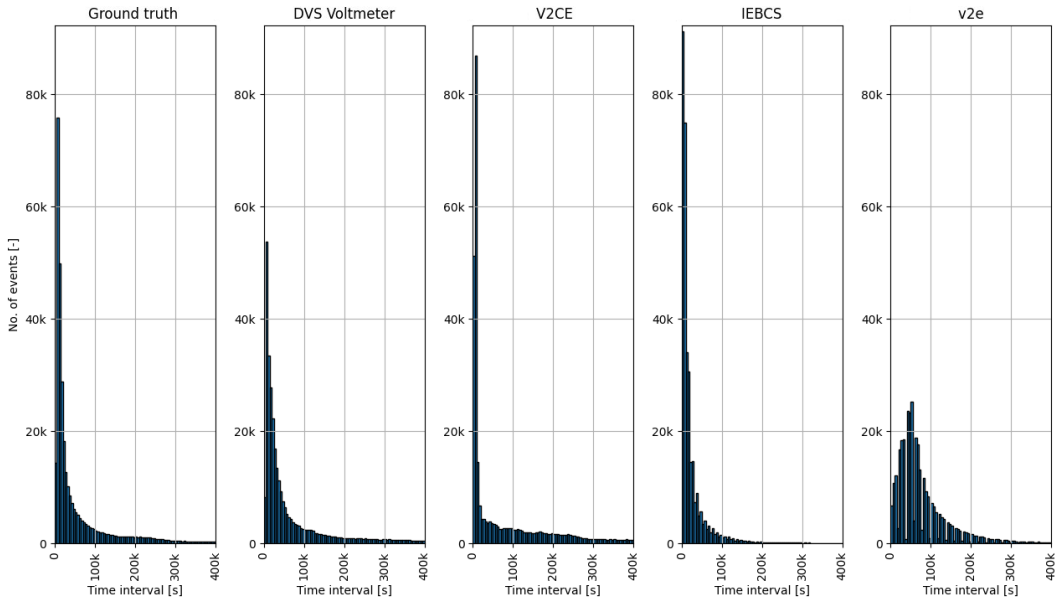



Figure 4.2: Histograms of time interval between consecutive events on a pixel for the ground truth and select event camera simulators evaluated on 400 thousand randomly chosen events.

### 4.1.5 Ease-of-use and flexibility

The v2e simulator is a rather mature software with good documentation which is ready to be used in a pipeline. The calibration however may be extremely tedious.

V2CE is by far the easiest simulator to use if the user mainly wants high fidelity simulation of an event camera-like sensor and they are willing to use pre-trained neural networks supplied by the authors of the software. However, if the user wants to simulate a specific sensor under specific condition, V2CE on the other hand becomes the most complicated option as training the network is challenging and requires substantial amounts of data.

IEBCS proved to be by far the worst simulator in terms of ease-of-use. The software is not ready to be used in a pipeline and the user is required to spend a lot of time learning how to use it from poorly documented examples. It is also really difficult to tune well and the user is constantly faced with significant amount of artifacts in the resultant simulations.

DVS-Voltmeter is relatively ill-prepared for use in a pipeline and any user who wants to utilize it is required to write significant amount of functions on their own. With regards to tuning, it is extremely flexible, which comes as a double edged feature. It is possible to obtain very precise simulations, but there is a high risk of overfitting. The authors have supplied a calibration procedure for DAVIS cameras and it brings decent results. On the other hand it is very laborious to implement and manually tune. Unlike in v2e or IEBCS there is no clear physical meaning to the tuning parameters and that makes the process much more difficult.

### 4.1.6 Trade-off between event camera simulators

The results of the testing can be summarized in a short trade-off addressing the relevant metrics which is shown in Table 4.3. As it can be seen v2e and V2CE scored the same amount of points. Preference was given to v2e because of the fast time-frame of the research.

## 4.2 Event camera model

This section will describe the event camera model used in the simulations. First it will introduce the simulated non-ideal behavior and secondly, it will discuss the limitations of the event camera model.

| Simulator | Comp. cost | Timestamp fidelity | Event c. and distr. | Ease-of-use and flex. |
|---|---|---|---|---|
| v2e | + | - | + | ++ |
| DVS Voltmeter | + | ++ | + | - |
| V2CE | - | + | ++ | + |
| IEBCS | + | + | – | - |
| EventGAN | + | *not tested* | *not tested* | *not tested* |

Table 4.3: Trade-off table assessing qualities of different event camera simulators. '++' translates as excellent, '+' as good, '-' as poor, and '–' as insufficient.

### 4.2.1 Simulated non-ideal behaviors

Brief introduction to non-idealities tied to event camera pixel can be found in Section A.5. Of these effects, it was decided to simulate threshold value variation, leakage currents and the low-pass filter-like behavior of the pixel as they are expected to be most significant in a mostly well-lit scene. An assumption is made that the pixels behave similarly to the pixels in the bright recordings Neuromorphic-Caltech101 dataset (Orchard et al. (2015)), which have their non-ideal behavior described in work of Hu et al. (2020). Although this assumption is possibly far-fetched, it is necessary due to lack of better data. The values are summarized in Table 4.2.1.

| Condition | Value |
|---|---|
| Threshold variation | 0.03 |
| Leak events frequency | 0.1 Hz |
| Cutoff frequency | 200 Hz |

Table 4.4: Values for event camera simulator non-ideality parameters, data based on Hu et al. (2020)

A decision was made not to simulate shot noise, which is suggested by Hu et al. (2020) to be negligible in these conditions, and pixel refractory period which seemed to have a rather limited effect on the number of events produced by each pixel during experiments.

### 4.2.2 Model limitations

As Lin et al. (2022) notes, the v2e simulator does not accurately distribute timestamps at resolutions much smaller than the input frame rates, which in case of this work is 100 frames per second as previously mentioned. Instead, it tends to chunk the events around discrete times between the supplied frames as shown in Figure 4.1. Hence, if accumulation of events into frames is used, the v2e input frame rate serves as a minimum resolution.

Furthermore, as already mentioned in Section 3.3, v2e reacts very poorly to the undesirable popping effects coming from use of a 3D model with discrete level of detail texturing. The sudden appearances lead to spikes in events in places where there should be none. However, as Figure 4.3 shows, for a typical trajectory, there is one such spike per 200 or 300 frames, which is deemed fully acceptable for learning.

Lastly, it is acknowledged that it was observed that with increasing the v2e leak frequency settings, some pixels become inactive with time. This lead to choice of leak frequency value on a lower side of the realistic and further investigation in this area is recommended. The inactive pixels are extremely rare though and they are believed to not significantly affect the processing.

## 4.3 Validation procedure

This section will first discuss the choice of the ground truth data to validate the dataset. Then it will continue to introduce the validation metrics. Lastly, the processing of event data for the validation and the validation process itself will be shown.

### 4.3.1 Choice of ground truth data

There is a number of vehicle motion event-based datasets recorded using an actual event camera. Currently, the most widespread ones are the Multi Vehicle Stereo Event Camera Dataset (MVSEC) by Zhu, Thakur, et al. (2018) and Stereo Event Camera Dataset for Driving Scenarios (DSEC) by M. Gehrig, Aarents, et al. (2021).

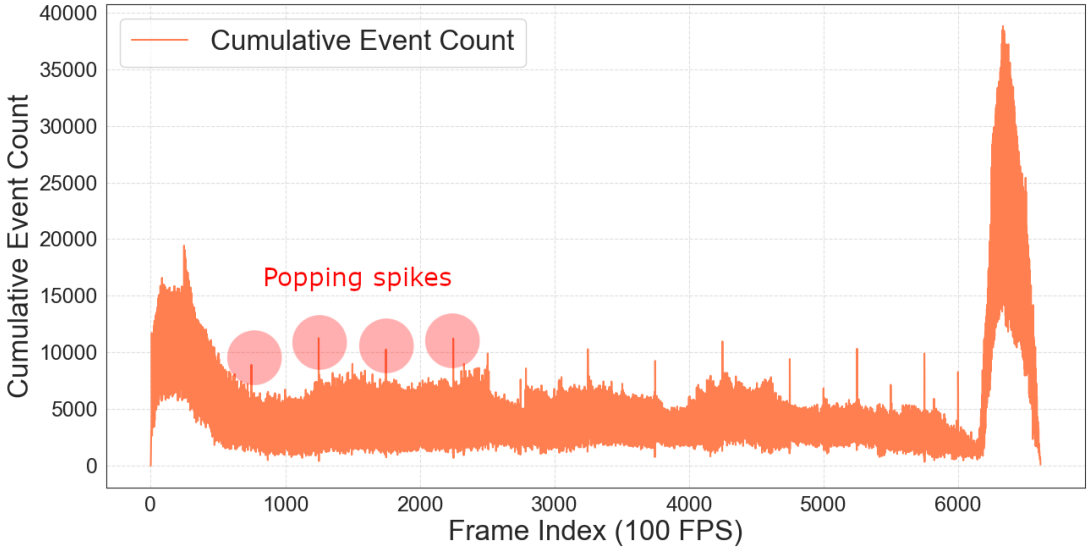



Figure 4.3: Spikes in event count per input frame caused by the discrete level of detail changes in the underlying surface model. Notice that they spikes occur only once per 200 to 300 frames

However, neither of these two commonly used datasets can be trusted to accurately represent the extremely light and dark regions seen during a lunar landing. It can also hint at nothing about the interaction between the event camera pixel and the often ragged lunar surface. Luckily, there exist two purpose-built datasets for planetary landing - ventral landing dataset by McLeod et al. (2022) and Synthetic Lunar Terrain dataset by Märtens et al. (2024).

As elaborated later in the Subsection 6.1.1 the camera will produce the same output if it is moving with a velocity $\mathbf{v}$ towards an object at distance $d$ and when the velocity is $\frac{\mathbf{v}}{k}$ and the distance $\frac{\mathbf{d}}{k}$ with $k$ being any positive real number. This allows to partially recreate the landing conditions in the constrained conditions of a laboratory. Both the dataset by McLeod et al. (2022) and the dataset by Märtens et al. (2024) use this strategy to obtain their recordings.

The dataset by McLeod et al. (2022) uses a robotic arm with a depth sensor attached, which moves around 2D and 3D-printed planetary surfaces. As all the trajectories are purely ventral, this is enough to fully reconstruct the optical flow as well, using the Equation 3.4. Unluckily, the surfaces used in this dataset represent primarily Mars and Mercury, which both differ from the Moon significantly in terms of surface structure.

On the other hand the dataset of Märtens et al. (2024) uses an arm manually moved above a large 3D model of terrain near the Lunar South Pole. This is interesting as with the interplay of very strong shadows and light in this region, it is a great target for optical navigation. There are no data provided, useful to determine the optical flow. Based on the similarity, a choice was made to proceed with the dataset of Märtens et al. (2024).

### 4.3.2 Validation metrics

Validating event camera models is challenging, because it is notoriously difficult to "match" event clouds to each other due to a lack of clear metrics as previously mentioned. Hence, it is necessary to look at the relevant use scenario for the respective event stream and pick metrics for the purpose. This section will outline the choices made in this work and motivate them appropriately.

With regards to the use of neural networks for optical flow determination, this is a relatively easy task. Vast majority of such neural networks uses accumulated events frames with a predefined time resolution as an input. As an example we can list the networks of C. Lee et al. (2020), Zhu, Yuan, et al. (2018) and Kosta and Roy (2023). Due to this accumulation step, the exact timing information for the events is not very important as it is lost anyway. What is critical though is the event count and distribution within this respective frame.

Moreover, we do not have to focus on the entire event stream, but we can only look at the spatio-temporal areas with high event rate triggered by edges. Such areas carry by far the most flow information and hence they are

the most important for the purpose. The next few paragraphs will define a method allowing to extract mean event rate $\mu_{ev}$ and its variance $\sigma_{ev}$ to describe event count and distribution in these spatio-temporal regions.

Let us define a set of events $\mathcal{E}$ which belongs to a discrete time step $\Delta t$.

$$\mathcal{E} = \{e(\hat{x}_i, \hat{y}_i, p_i, t_i) \mid t_1 \leq t_i \leq t_1 + \Delta t\} \tag{4.2}$$

Every element of this set can be accumulated into a two-dimensional event frame $F_{ev}$ as follows with $\delta$ representing the Kronecker delta.

$$F_{ev}(\hat{x}, \hat{y}) = \sum_{e_i \in \mathcal{E}} \delta(\hat{x} - \hat{x}_i)\delta(\hat{y} - \hat{y}_i) \tag{4.3}$$

The frame $F_{ev}$ is filtered into a set of pixels $\mathcal{P}_{ev}$, which includes only edge pixels with higher event rate, using a uniform kernel $K_{ev}$ with $a_{ev}$ being a threshold number.

$$\mathcal{P}_{ev} = \{F_{ev}(\hat{x}, \hat{y}) \mid F_{ev}(\hat{x}, \hat{y}) * K_{ev} \geq a_{ev}\} \tag{4.4}$$

From the set $\mathcal{P}_{ev}$ we can define our two crucial metrics which describe the edge behavior. That is the mean event rate $\mu_{ev}$ and its variance $\sigma_{ev}$.

$$\mu_{ev} = \frac{1}{|\mathcal{P}_{ev}|} \sum_{P_i \in \mathcal{P}_{ev}} P_i \tag{4.5}$$

$$\sigma_{ev}^2 = \frac{1}{|\mathcal{P}_{ev}|} \sum_{P_i \in \mathcal{P}_{ev}} (P_i - \mu_{ev})^2 \tag{4.6}$$

It is noted that both $\mu_{ev}$ and $\sigma_{ev}$ are dependent on the event camera parameters or the parameters of its model. They also naturally depend on the texture of the surface, which is being recorded by the event camera, and the optical flow.

### 4.3.3 Processing of event data

On the ground truth side, 10 random recordings from the SLT dataset were chosen and they were processed in a following fashion.

1. Events were accumulated into frames $F_{ev}$ with a 25 Hz frequency
2. Frame-based Lucas-Kanade algorithm was applied to a grayscale representation of the flattened frames to find the optical flow magnitude
3. Metrics $\mu_{ev}$ and $\sigma_{ev}$ were calculated from the frames and their relation to the optical flow was established

The choice of the frame-based Lucas-Kanade algorithm to extract the optical flow from the events may be very surprising. In this case, it has its logic though. The dataset contains areas of both extreme darkness and light. The areas of extreme darkness in the craters are almost event free as a result. Hence, the edges of the craters are very well visible to such an algorithm and result in reliable predictions. Moreover, it is very easy to visually verify that the Lucas-Kanade algorithm indeed tracks the target points correctly and its implementation with the OpenCV library by Bradski (2000) is trivial.

On the side of the synthetic data a simple translational trajectory with variation of optical flow is prepared and processed in an identical fashion.

### 4.3.4 Tuning and results

The proposed tuning process is iterative. During the first iteration, the data from the SLT dataset are processed according to the steps described in the previous subsection and an initial camera threshold $\theta_{ON/OFF}$ value is chosen for the event camera simulator based on the settings used for the SLT dataset. $\mu_{ev}$ and $\sigma_{ev}$ are found for both SLT and the synthetic data and compared against the optical flow. To tweak $\mu_{ev}$ for the next iteration, it is recommended to change the threshold $\theta_{ON/OFF}$ for the v2e simulations. On the other hand, to have a major effect on $\sigma_{ev}$, it is recommended to decrease contrast in the PANGU-generated frames using Tomar (2006). The flow of the process is illustrated in Figure 4.4.

For the purposes of this work, it was considered unnecessary to try to make the tuning process automatic and the threshold and contrast values for further iterations were simply based on educated guessing. In this

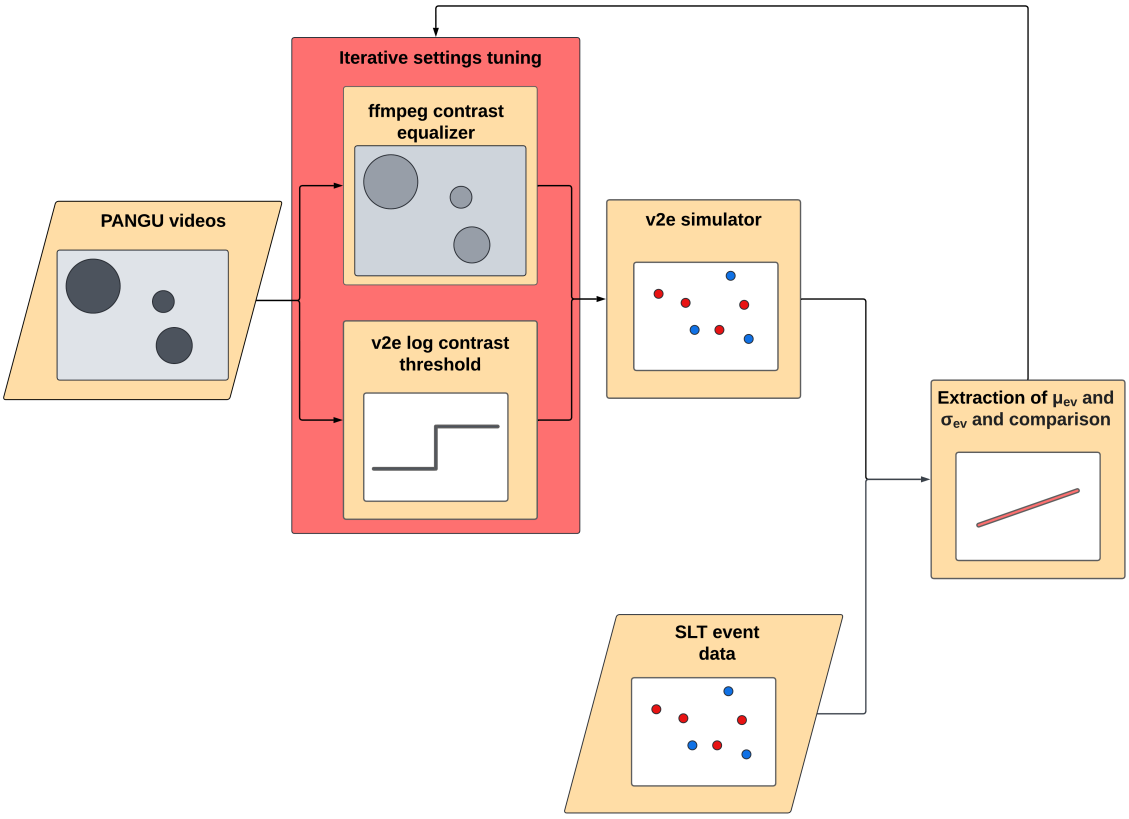



Figure 4.4: Iterative tuning process

case it seems sufficient as it is not the task to fully restore the, anyway very variable, camera settings like the thresholds, but to merely create a roughly realistic model of the pixel-surface interaction.

The tuning in this work converged to what was deemed acceptable after 16 iterations. The graphs representing the variation of $\mu_{ev}$ and $\sigma_{ev}$ against the magnitude of optical flow are shown in the Figure 4.5. The final logarithmic threshold $\theta_{ON}$, $\theta_{OFF}$ value used in the work is set to 0.12 and the contrast was dimmed by 28 % compared to original PANGU imagery.

Lastly, based on the personal discussion with the authors of Märtens et al. (2024), it is noted that even the synthetic surface used for their recordings, differs from the actual lunar surface. Due to health and cost issues, their work uses a combination of crusher dust and blast furnace slag to model the terrain leading to an albedo very close to lunar south pole. However, such material leads to a more grainy texture than an actual regolith. As such, the event camera signal becomes more spread around the edges, compared to sharper transitions expected in reality. Nevertheless, this work does not attempt to mitigate this issue. It merely states that the assumption to validate against the data from Märtens et al. (2024) is conservative as in reality the edges are expected to be sharper and hence produce more orderly events and be generally easier to work with.

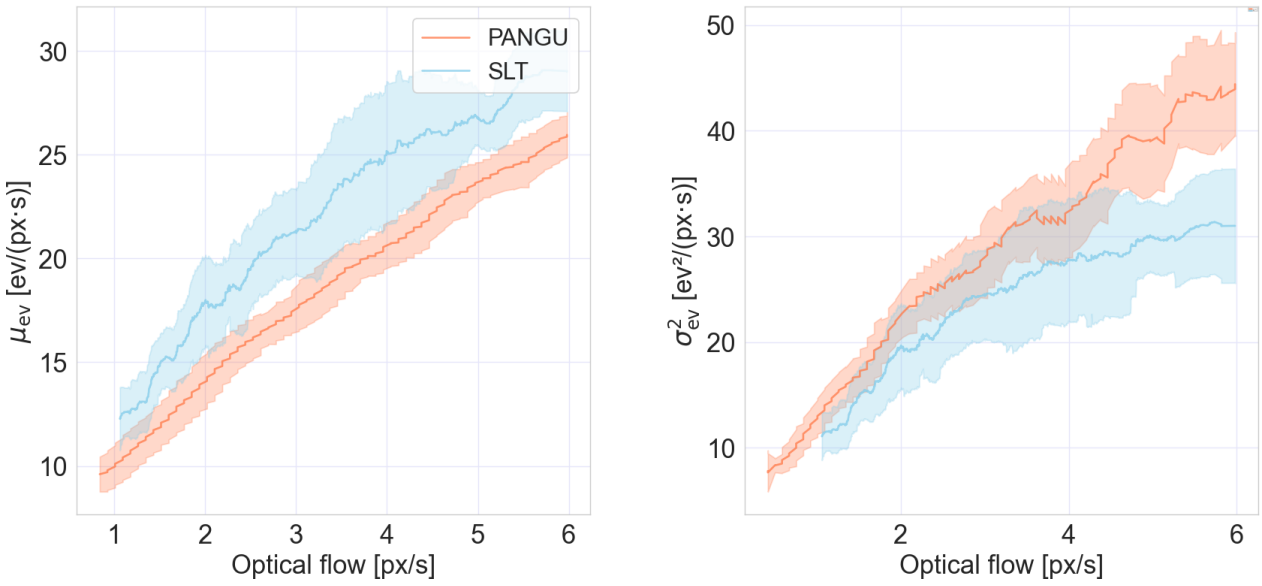



Figure 4.5: Event rate and its variance during the validation process. SLT represents the ground truth and PANGU the synthetic imagery. Shaded area belongs to one standard deviation.

# Part II

# Learning optical flow and estimating egomotion

## Overview

This part of the report will introduce the process of learning optical flow using spiking neural networks from the dataset and recovery of egomotion from the optical flow. Figure 4.6 gives a rough outline of the egomotion inference task. First, optical flow is found from events. The resultant flow is inputted into an egomotion estimator, which will give the final product with assistance of additional information about altitude and possibly attitude.

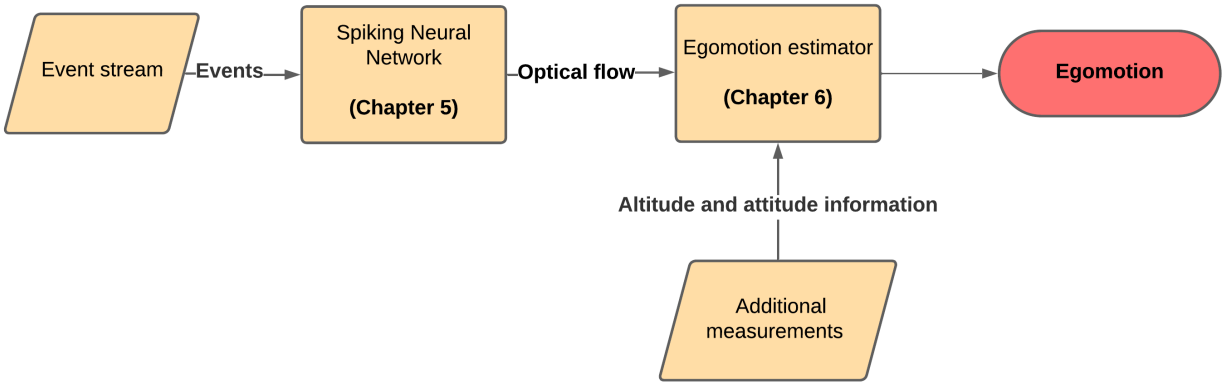



Figure 4.6: Overview of the process of learning the optical flow and egomotion inference

Chapter 5 will focus on the supervised learning of the optical flow from events. First, it will discuss the choice of the spiking neural network models for this purpose. Then it will introduce architectures, training strategies and handling of temporal information inside these chosen networks. It will add a discussion about the loss functions and lastly present the results of the training.

Chapter 6 will study the problem of turning optical flow into egomotion. First, it will properly introduce the problem. Secondly, it will introduce three regimes of the aforementioned additional attitude and altitude information which are to be studied. Then, it will focus on solving the problem from mathematical perspective. Propagation of errors and effect of spatial correlation between errors in the optical flow will be studied and a simple navigational filter will be laid out. Lastly, results of simulations will be presented.

# Chapter 5

# Learning optical flow using spiking neural networks

This chapter will deal with the task of learning the optical flow from events using spiking neural networks. First, it will discuss the choice of two models for further evaluation in Section 5.1. Afterwards, it will introduce the architectures of the models in Section 5.2. Furthermore, the handling of temporal information in the models, an area of great importance and potential in spiking neural networks, will be compared in Section 5.3. Afterwards, loss functions will be discussed, involving a proposed anti-spatial correlation term in Section 5.4. A benchmark model is introduced in Section 5.5. Lastly, the results will be presented and considered in Section 5.6.

## 5.1 Choice of the SNN models for evaluation

The literature study identified a number of possible candidate models for the spiking neural network, which greatly vary in architecture, number of parameters and purpose. While it is very difficult to make a fair comparison, perhaps the best available metric is the performance on the indoor trajectories of the MVSEC dataset, which is reported for all the considered networks. The MVSEC dataset performance for various identified SNNs shown in Table 5.1 reveals a remarkable dominance of the OF-EV-SNN by Cuadrado et al. (2023). The network in question achieves superior performance with a significantly lower amount of parameters than is common. Considering that OF-EV-SNN is also designed to be particularly friendly to neuromorphic hardware, it is an obvious choice for testing.

However, to show that this counterintuitive result is true and the OF-EV-SNN will outperform the other networks on the landing task as well, it is necessary to also test at least one of the competing networks to confirm that this disparity is present. For this purpose, the Micro Adaptive-SpikeNet was chosen for its similar architecture and size, which allows a very interesting comparison of two different ways how to handle temporal information.

| **Model** | **Author** | **Indoor1** | **Indoor2** | **Indoor3** | **Sum** | **Params** ($\times 10^6$) |
|---|---|---|---|---|---|---|
| LIF-EV-FlowNet | Hagenaars et al. (2021) | 0.71 | 1.44 | 1.16 | 3.31 | 20.4 |
| SNN-Timelens | Schnider et al. (2023) | 0.70 | 1.30 | 1.05 | 3.05 | 25.35 |
| Base AdaptiveSpikeNet | Kosta and Roy (2023) | 0.84 | 1.59 | 1.36 | 3.79 | 13.04 |
| Micro AdaptiveSpikeNet | Kosta and Roy (2023) | 0.95 | 1.74 | 1.48 | 4.17 | **0.93** |
| OF-EV-SNN | Cuadrado et al. (2023) | **0.58** | **0.62** | **0.67** | **1.87** | 1.22 |

Table 5.1: Average endpoint error performance comparison of various well-performing models on the indoor trajectories of the MVSEC dataset. The data is taken from the respective articles.

## 5.2 Network architecture and training

This section will describe architecture and training methods present in the chosen neural networks. It relies on Cuadrado et al. (2023) and Kosta and Roy (2023) for this information.

Both OF-EV-SNN and Adaptive-SpikeNet utilize the so-called U-Net architecture, described in Appendix D, with a four-stage decoder and encoder. This high-level similarity allows for an interesting comparison between

these two networks as it makes it possible to assess the performance of lower-level components. A high-level illustration of this architecture, valid for both networks, can be seen in Figure 5.1.

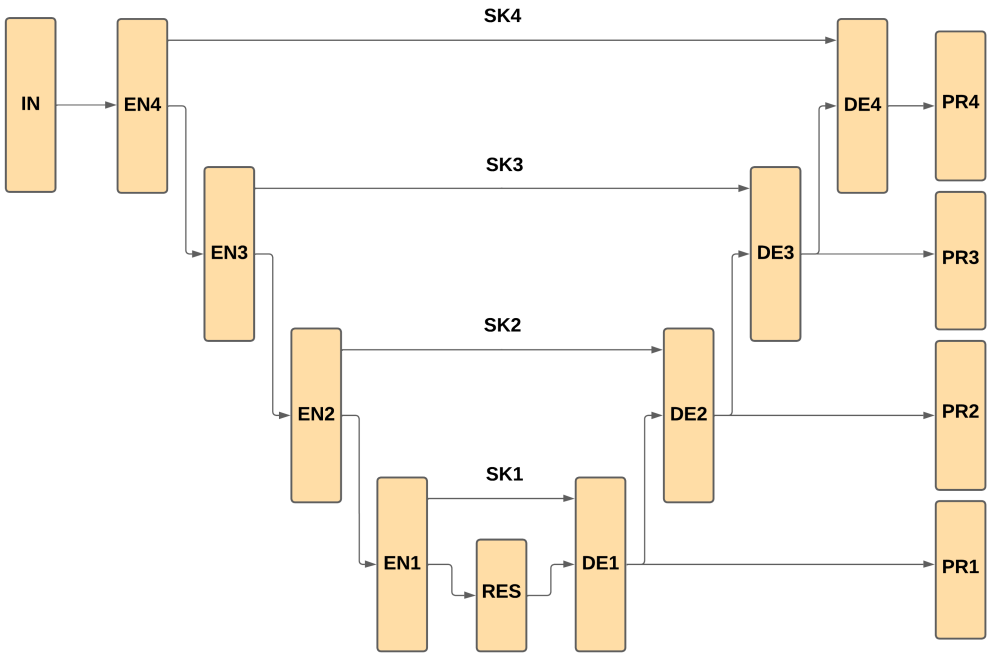



Figure 5.1: U-Net-like architecture of the chosen networks

Both of the networks take the input (further described in Subsection 5.3.1) into an input block (marked IN), which is connected to a four-stage encoder (EN1-EN4). Each of the encoder stages performs a convolution and doubles the number of channels. However, as Cuadrado et al. (2023) highlights, OF-EV-SNN chooses maximum pooling instead of strided convolution for improved neuromorphic hardware-friendliness in contrast to Adaptive-SpikeNet. It is also important to note that the OF-EV-SNN encoder performs convolution along temporal channels as well to improve performance as elaborated in Subsection 5.3.2.

OF-EV-SNN offers a single residual layer (RES) with non-strided convolution to process the encoder input. On the other hand, a substantial part of the Adaptive-SpikeNet parameters belongs to a residual with four layers and two internal skip connections to ensure preservation of undistorted information.

Both of the networks utilize a four stage decoder as well (DE1-DE4). At each decoder stage the number of channels doubles with upsampling. Also, all of the stages are connected to the respective encoder stage by skip connections (SK1-SK4) in order to preserve low-level detail. The decoder stages output directly into predictor blocks (PR1-PR4).

A notable feature is that all the predictor blocks then play a role in the final inference which is at the same spatial resolution as the input block.

Both networks are trained using the, currently rather common, backpropagation with a surrogate gradient. This method is further described in Appendix C. Learnable leaks and thresholds, further described in Subsection 5.3.2, represent an interesting improvement of this method utilized in the Adaptive-SpikeNet network. While training times are highly dependent on exact formulation of the loss function, it can be generally expected that training a model will take a day or more on a modern GPU. The process is also highly challenging from perspective of resource management as training such networks with inherent recurrence is very memory intensive.

## 5.3 Handling of temporal event information

This section will describe the handling of temporal event information in the two chosen networks. It again relies on Cuadrado et al. (2023) and Kosta and Roy (2023) for this information.

One of the biggest challenges in event-based optical flow is appropriate utilization of event timing information, which can greatly improve performance. And the handling of temporal information in OF-EV-SNN and Adaptive-SpikeNet is in sharp contrast. OF-EV-SNN relies on temporal convolution to explicitly capture important relationships, while Adaptive-SpikeNet utilizes the inherent qualities of the neurons. This section will further discuss the differences in this part in the input to the network and then in the network itself.

### 5.3.1 Network input

Let $\mathcal{E}$ be a set of events between the times $t_1$ and $t_1 + \Delta t$.

$$\mathcal{E} = \{e(\hat{x}_i, \hat{y}_i, p_i, t_i) \mid t_1 \leq t_i \leq t_1 + \Delta t\} \tag{5.1}$$

Then Adaptive-SpikeNet uses the following accumulation scheme into an event frame with dimensions ($B$, $P$, $W$, $H$) where $B$ is the number of temporal bins, $P$ are the two polarity channels and $W$ and $H$ are width and height. First, event timestamps are converted to a range according to the number of bins as seen below.

$$b_i = (B-1)\frac{t_i - t_1}{\Delta t} \tag{5.2}$$

Then, the event frame $F_{ASN}$ is generated by bilinear interpolation along a temporal dimension kernel $K_{ASN}$ and assigning the events using Kronecker delta $\delta$ to a correct pixel.

$$F_{ASN}(\hat{x}, \hat{y}, p, b) = \sum_{e_i \in \mathcal{E}} K_{ASN}(b - b_i)\delta(p - p_i)\delta(\hat{x} - x_i)\delta(\hat{y} - y_i) \tag{5.3}$$

$$K_{ASN}(a) = \max(0, 1 - |a|) \tag{5.4}$$

The bilinear sampling allows to retain some of the temporal information as an event near an edge of a spatio-temporal voxel now also projects into the temporally neighboring voxel to a lesser extent. It is noted, that the Equation 5.3 slightly differs from the version presented in Kosta and Roy (2023) as the original work also carries out bilinear interpolation along the spatial dimensions. That is because events fall between pixels due to the rectification of the non-ideal camera. However, this can be ignored here as the pinhole camera used in this work is ideal, hence the use of Kronecker deltas instead.

The frame $F_{ASN}$ is further subdivided into a former and a latter group of events ($b < B/2$ and $b > B/2$) and according to polarity to form four channels. These four channels are then sequentially passed to the network over $B/2$ time steps. As Kosta and Roy (2023) notes, this allows to capture the small-scale temporal developments between each of the time steps, but also the large-scale temporal differences between the former and latter event group at the same time.

In the context of this work, $B = 10$ with 10 ms time steps leading to a 100 ms receptive field. A limitation is appreciated where the learning can be affected by the discrete effects from the v2e simulator which have a similar frequency. Please refer to Subsection 4.2.2 for further detail on these limitations. However, the bilinear interpolation process is believed to mitigate this issue partially. Values larger than 10 ms were not opted for as they tended to perform worse in training during short experiments.

On the other hand, OF-EV-SNN accumulates the events into a frame $F_{OES}$ with $B$ temporal bins as follows while $\Pi$ is the boxcar function. The frame $F_{OES}$ is then passed to the network at once.

$$F_{OES}(\hat{x}, \hat{y}, p, t) = \Pi(t - t_i)\delta(p - p_i)\delta(\hat{x} - x_i)\delta(\hat{y} - y_i) \tag{5.5}$$

$B = 21$ is used with the OF-EV-SNN in this work with each bin spanning 9 ms leading to 180 ms receptive field. Again, it is appreciated that this 9 ms is close to the minimum temporal resolution. However, models trained with such time spans again tended to perform better than with larger time spans based on short experiments and if there are discrete effects at play, they should disappear in the first layers with the temporal convolution described in the next section.

### 5.3.2 Neuron models and temporal convolution

The Adaptive-SpikeNet of Kosta and Roy (2023) relies purely on the inherent qualities of Leaky Integrate-and-Fire (LIF) neurons to learn the temporal dependencies from the provided input. The state equation of the LIF neuron taken from Kosta and Roy (2023) is shown below. There $\mathbf{u}_t^l$ is a vector of potential on neurons in layer $l$ at time step $t$, $\mathbf{W}_l$ the vector of synaptic weights between $l$ and the previous layer, $\mathbf{o}_t^l$ the vector of binary

spikes outgoing from the layer $l$ at time step $t$ and constants $\lambda_l$ and $v_{th}^l$ being the leak value and firing threshold respectively.

$$\mathbf{u}_t^l = \lambda_l \mathbf{u}_{t-1}^l + \mathbf{W}_l \mathbf{o}_t^{l-1} - v_{th}^l \mathbf{o}_t^l \tag{5.6}$$

Then, as Kosta and Roy (2023) notes, the generation of spikes $\mathbf{o}_t^l$ at layer $l$ and time step $t$ follows the rule presented in Equation 5.7.

$$\mathbf{z}_t^l = \frac{\mathbf{u}_t^l}{v_{th}^l} - 1, \quad \mathbf{o}_t^l = \begin{cases} 1, & \text{if } \mathbf{z}_t^l > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5.7}$$

A very interesting feature of Adaptive-SpikeNet are the layer-wise learnable values of $\lambda_l$ and $v_{th}^l$, which are incorporated in order to mitigate the well-known vanishing gradient problem mentioned by Hagenaars et al. (2021). This is in contrast with typical practice, where these values are set manually.

In contrast, OF-EV-SNN utilizes explicit temporal convolution for this task and the Integrate-and-Fire (IF) neurons used in the network are substantially less powerful than their LIF counterparts on their own. As shown in Equation 5.8, the IF neuron equation lacks the leak term and hence cannot prioritize more recent information. Spike firing follows the rule shown in the 5.7. Values of $\lambda_l$ and $v_{th}^l$ are hyperparameters to be manually tuned.

$$\mathbf{u}_t^l = \mathbf{W}_l \mathbf{o}_t^{l-1} - v_{th}^l \mathbf{o}_t^l \tag{5.8}$$

Surprisingly, OF-EV-SNN utilizes an unusual mechanism in which all neurons are reset after each forward pass. However, as Cuadrado et al. (2023) notes, this solution is compatible with neuromorphic hardware and even achieves lower power consumption values, because the reset is less demanding from an energy perspective than the leak.

The explicit temporal convolution in OF-EV-SNN is carried out with a kernel having a stride of one and a size of $l_{temp}$ along the temporal dimension while being unpadded. Hence, with every step of the encoder, the number of temporal bins decreases by $l_{temp} - 1$. As already noted in Subsection 5.3.1, the input event frame $F_{OES}$ has $B$ temporal bins and a $B$ is chosen such that the amount of bins at the residual layers is equal to one (hence, if there are e.g. five input layers and $l_{temp} = 5$, then $B = 21$ should be chosen).

As the temporal dimension disappears before reaching the decoder, this raises an obvious question of how the skip connections are realized with the respective decoder and encoder stages having different dimensions. Here Cuadrado et al. (2023) proposes to prioritize the most recent information and connect only the last temporal bin using the skip connection, disregarding the rest.

## 5.4 Loss functions

The process of determining the loss functions was iterative and to a large extent depended on the analysis in Section 6.3. The focus was to provide such optical flow estimate, that will behave well if used as an input to the egomotion estimator described in Section 6.2.

Based on the roughly linear relationship between Average Endpoint Error (AEE) and the errors in egomotion estimation as seen in Figure 6.4, it was decided to use simple AEE as loss type $\mathcal{L}_a$ for training as seen in Equation 5.9 with $\mathcal{P}_{im}$ being set containing coordinates of all pixels in the image plane with side $W$.

$$\mathcal{L}_a = \frac{1}{W^2} \sum_{\hat{x},\hat{y} \in \mathcal{P}_{im}} \|\hat{\mathbf{u}}_{\text{gt}}(\hat{x},\hat{y}) - \hat{\mathbf{u}}_{\text{est}}(\hat{x},\hat{y})\| \tag{5.9}$$

However, as explained in Section 6.3, large-scale overfitting to global context can have major negative effects on the egomotion estimations. Hence, a loss term penalizing such behavior is proposed. Geometric mean filter of size $2l_m + 1$ is passed over both components of the flow field difference $\hat{\mathbf{u}}_{\text{gt}} - \hat{\mathbf{u}}_{\text{est}}$ to form vector $\mathbf{m}(\hat{x},\hat{y})$ as shown in Equation 5.10. This vector should tend to zero if there are no spatial correlations between the error of different point inferences. Hence, it is summed over the field to form the loss term $AC$ as shown in Equation 5.11. Value of $2l_m + 1 = 49$ was used in for this entire work based on experimentation.

$$\mathbf{m}(x,y) = \frac{1}{(2m+1)^2} \sum_{i=-l_m}^{l_m} \sum_{j=-l_m}^{l_m} (\hat{\mathbf{u}}_{\text{gt}}(x+i,y+j) - \hat{\mathbf{u}}_{\text{est}}(x+i,y+j)) \tag{5.10}$$

$$\mathrm{AC} = \frac{1}{W^2} \sum_{\hat{x},\hat{y} \in \mathcal{P}_{im}} \|\mathbf{m}(\hat{x},\hat{y})\| \tag{5.11}$$

The correlation-aware loss type is denoted as $\mathcal{L}_b$ and takes the form shown below. Experience had shown that the value of constant $\rho = 0.5$ leads to decent results.

$$\mathcal{L}_b = \mathrm{AEE} + \rho \mathrm{AC} \tag{5.12}$$

Overall, four models with different loss terms were trained. They all are listed below together with colors, which are used for these models in all the figures in this report:

- ASN model uses $\mathcal{L}_a$ loss and will be signified by this color:
- ASN AC model uses $\mathcal{L}_b$ loss and will be signified by this color:
- OES model uses $\mathcal{L}_a$ loss and will be signified by this color:
- OES AC model uses $\mathcal{L}_b$ loss and will be signified by this color:

## 5.5 Benchmark

Finding a fair benchmark to the spiking neural networks is difficult. For a real-time navigation application, computational cost matters tremendously if a computer with Von Neumann architecture is used. However, with a spiking neural network running on a neuromorphic chip, this is much less of an issue. To keep the comparison meaningful, it was decided to pick a close counterpart of the two spiking neural networks, which has state-of-the-art or near-state-of-the-art performance.

And the closest such analogue is the hybrid Spike-FlowNet of C. Lee et al. (2020) which combines a spiking and analog neural network and follows the same four-stage U-Net architecture as OF-EV-SNN and Micro Adaptive-SpikeNet. While this model is reported to achieve a somewhat worse MVSEC performance than e.g. E-RAFT by D. Gehrig et al. (2018), it is near the state-of-the-art. The amount of channels in the encoder and decoder can be reduced in order to match the fully spiking models. The benchmark will be from now on referred to as the BM model.

As C. Lee et al. (2020) explains, the idea behind the Spike-FlowNet model is fairly straightforward. The spiking encoder handles the temporal information in the input, but the analog decoder ensures that no malicious effects of spike vanishing appear. Diagram of the architecture together with further explanation can be found in Section D.3.2.

BM model uses $\mathcal{L}_a$ loss and will be signified by this color:

## 5.6 Experiments and results

This section will report on the setup of experiments and the respective results for use of both the Micro Adaptive-SpikeNet and the OF-EV-SNN spiking neural networks in learning optical flow. First, it will introduce the parameters and methods used during the training. Then it will interpret the results of said training.

### 5.6.1 Implementation

Both Micro Adaptive-SpikeNet and Spike-FlowNet were based on modification of code provided by Ponghiran et al. (2023). This implementation utilizes pure PyTorch of Paszke et al. (2019) with custom spiking modules by the authors.

OF-EV-SNN network was adapted from code of Cuadrado et al. (2023). Here, PyTorch in combination with SpikingJelly library by Fang et al. (2023) is used.

### 5.6.2 Method of training

As the event rate and structure closely resembles the DSEC or MVSEC dataset, original hyperparameters recommended by the authors of Kosta and Roy (2023) and Cuadrado et al. (2023) were used to train the networks. For Micro Adaptive-SpikeNet that means fixed learning rate of $1e^{-4}$ and a batch size of 8. For OF-EV-SNN it is learning rate of $2e^{-4}$ which decreases by half every decade of epochs. Batch size of 8 was also used. Lastly, for the benchmark, learning rate of $5e^{-}5$ was utilized as recommended by C. Lee et al. (2020).

During training, no augmentation was used. Based on preliminary experiments with cropping, flipping, rotation and event dropout, it is believed that it would result in an improvement in performance. However, such a result would be very specific to the dataset used and there is no guarantee that it would work similarly with another landing dataset. Moreover, while the absolute values of performance slightly improve, the relative standing of the models was found to be the same.

The original test set was further split into a test and validation set. The models were trained until the loss on the validation set stopped decreasing, which happened after 30-40 epochs for all of them. The model corresponding to the epoch with best results on the validation set was then picked.

### 5.6.3 Average Endpoint Error

Table 5.2 reports the mean AEE values for each of the models. First of all, it should be noted that the values significantly exceed the AEE values reported for the MVSEC dataset in Table 5.1. This suggests that the size of the dataset, lack of variety or the nature of the flow makes it significantly harder to estimate than in MVSEC. It remains for future investigations to explain this disparity. Secondly, it can be seen that the best performing model is the OES AC by a considerable margin of 15 % ahead of OES and with 13 % on the benchmark. Both the ASN based models trail behind the rest significantly, albeit not as significantly as Table 5.1 implies for MVSEC. However, the OF-EV-SNN superiority can be considered confirmed.

| **Model** | **V** | **N** | **D** | **Sum** |
|---|---|---|---|---|
| ASN | 1.31 | 4.03 | 3.76 | 9.10 |
| ASN AC | 1.34 | 3.72 | 3.63 | 8.69 |
| OES | **0.75** | 2.78 | 3.26 | 6.79 |
| OES AC | 1.03 | **1.76** | **2.93** | **5.72** |
| BM | 0.70 | 3.01 | 2.90 | 6.61 |

Table 5.2: Mean AEE values for models across trajectory types (V, N, D) and their totals.

For better illustration, the median values AEE together with the distributions for different models and trajectory types are also shown in Figure 5.2. On the first glance it can be seen that the median AEE is substantially different from the mean AEE for almost all of the models and trajectories. Another noteworthy fact is the marginal difference between ASN and ASN AC in contrast to significant differences between OES and OES AC where the latter mostly outperforms the former.

It is an interesting question whether AEE increases in fields with a higher average magnitude of the optical flow. If that is true, trajectories with significant attitude maneuvers like divert trajectories will always lead to worse AEE values as they typically involve optical flow with large magnitudes. Figure 5.3 reveals that there exists such a relationship and higher optical flow leads to increased AEE with both the OES AC and the BM model. A question whether this is due to inherently higher difficulty of predicting the larger optical flow vectors with the same precision or due to underrepresentation of this condition in the dataset remains to be answered in future work.

### 5.6.4 Spatial error correlation

In Section 5.4, a loss term AC is proposed, which is supposed to discourage large scale spatial correlation between errors and as such improve the performance on egomotion estimation task. This loss term was evaluated on all the five models and three trajectory types. The median values together with the distributions are shown in Figure 5.4.

It is noted that the overall values of AC are very high and nearing the values of AEE in almost all models. This suggests that the models still overvalue global context over local context highly, and this is adversely

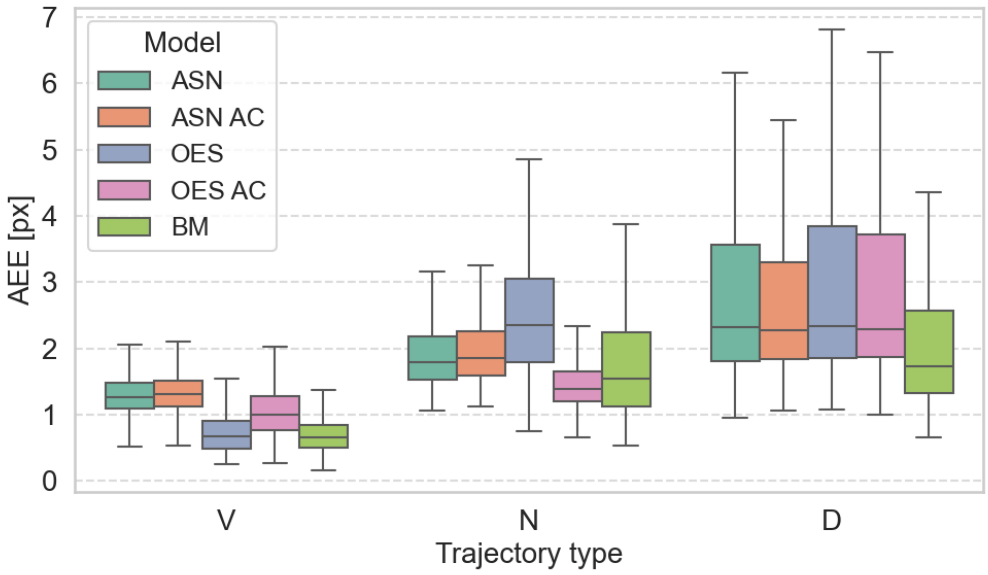



Figure 5.2: Distribution of AEE for the five different models and three different trajectory types (ventral - V, nominal - N and divert - D).

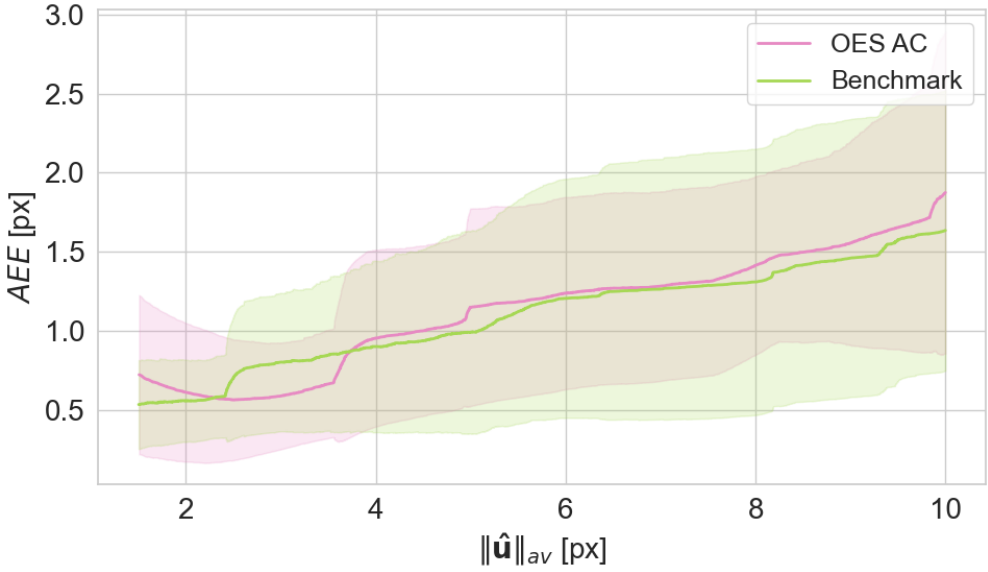



Figure 5.3: Average Endpoint Error as a function of average optical flow in OES AC and BM models. Shaded area represents one standard deviation.

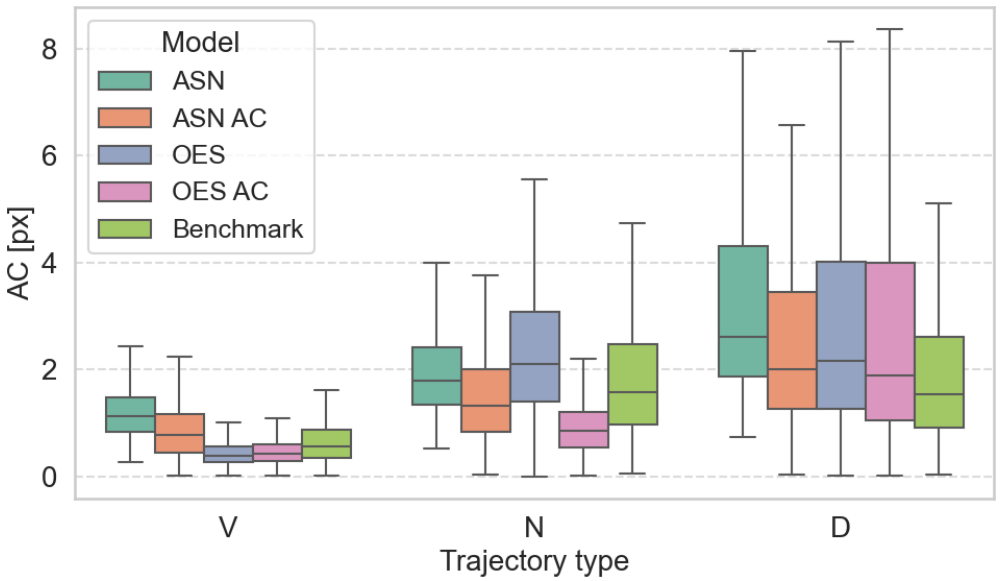



Figure 5.4: The distribution of Anti-Correlation term AC among the five models and three trajectory types. Notice the superiority of the AC models and high absolute values.

impacting the results. However, it can be also seen that applying the AC loss term during training has strong positive aspects in this area and preliminary experiments with more complex functions which punish correlation at multiple levels of scale suggest possible further improvements.

# Chapter 6

# Optical flow to egomotion

This chapter will treat the problem of estimating optical egomotion from optical flow. First, the problem will be formally introduced, and three regimes with varying input variables will be discussed in Section 6.1. Secondly, it will be shown how an overdetermined system appears during the estimation process and its solution will be shown in Section 6.2. Then a Monte Carlo sensitivity analysis focused on propagation of errors and effects of error correlation follows in Section 6.3. Also included is a short look at a navigation filter, which increases performance of the estimator in Section 6.4.

## 6.1 Problem overview

The first part of this section will introduce the problem at hand, establish error metrics and give a rough outline the solution process. It will also show which variables are necessary to uniquely solve the problem. The second part of this section will outline and justify three different test regimes of varying complexity used in this work, which all utilize different set of known variables.

### 6.1.1 Problem introduction

The purpose is to estimate the vehicle velocity $\mathbf{v} = [v_x, v_y, v_z]^T$ in the inertial frame $\mathcal{F}_i$ ($O_i$, $x$, $y$, $z$) from a set of multiple optical flow measurements $\{\hat{\mathbf{u}}(\hat{x}, \hat{y}) \mid \hat{x}, \hat{y} \in \mathcal{P}\}$ using a planar approximation for the lunar surface. In order to achieve this, it is necessary to first use the optical flow to estimate the vehicle velocity $\mathbf{v_b} = [v_{b,X}, v_{b,Y}, v_{b,Z}]^T$ in the non-inertial vehicle body frame $\mathcal{F}_b$ ($O_b$, $X$, $Y$, $Z$) and, if attitude is not considered known, a vector $\mathbf{n} = [n_X, n_Y, n_Z]^T$ which is normal to the planar approximation of the lunar surface. This vector can then serve to find a transformation between the two reference frames. For better illustration of the coordinate systems, please refer to Subsection 2.1.1.

It is noted that such a problem is ill-posed without additional information. As Grabe et al. (2015) notes, there is an inherent scale ambiguity, in the estimation of velocity using optical flow. The optical flow $\hat{\mathbf{u}}$ in point $\hat{x}, \hat{y}$ can be expressed using Equation 6.1 with $Z$ being the depth of a scene point, which projects into point $\hat{x}$, $\hat{y}$ on the image plane.

$$\hat{\mathbf{u}}(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = \mathbf{K}(\hat{x}, \hat{y})\boldsymbol{\omega_b} + \frac{1}{Z}\mathbf{L}(\hat{x}, \hat{y})\mathbf{v_b} \tag{6.1}$$

The rotation-related matrix $\mathbf{K}(\hat{x}, \hat{y})$ and translation-related matrix $\mathbf{L}(\hat{x}, \hat{y})$ in Equation 6.1 can be expressed as follows.

$$\mathbf{K}(\hat{x}, \hat{y}) = \begin{bmatrix} -\hat{x}\hat{y}/f & (f^2\hat{x} + \hat{x}^2)/f & -\hat{y} \\ -(f^2\hat{y} - \hat{y}^2)/f & \hat{x}\hat{y}/f & \hat{x} \end{bmatrix} \tag{6.2}$$

$$\mathbf{L}(\hat{x}, \hat{y}) = \begin{bmatrix} f & 0 & -\hat{x} \\ 0 & f & -\hat{y} \end{bmatrix} \tag{6.3}$$

The depth of the scene point $Z$ is an unknown variable, but it can be constrained using the planar surface assumption shown in Equation 6.4 as P. Li et al. (2015) notes.

$$n_X X + n_Y Y + n_Z Z = -z \tag{6.4}$$

By studying Equation 6.1 and Equation 6.4, we can observe that the optical flow will be identical if $\mathbf{v_b}, -z$ is a solution, but also if $k\mathbf{v_b}, -z/k$ is a solution, with $k$ being a scaling factor. Hence, as Grabe et al. (2015) notes, another metric measurement from a different instrument needs to be provided to uniquely reconstruct $\mathbf{v_b}$. Additionally, as Grabe et al. (2015) also states, set of at least four optical flow measurements is required, so that a system of stacked instances of Equation 6.1 can have a unique solution. This number decreases to three if $\boldsymbol{\omega_b}$ is assumed to be known.

Going further, as Azzalini et al. (2023) states, after the velocity $\mathbf{v_b}$ and vector $\mathbf{n}$ are found they can be converted to $\mathbf{v}$ in the inertial frame as shown below.

$$\mathbf{v} = \mathbf{R}(\phi, \theta, \psi)\mathbf{v_b} \tag{6.5}$$

With the matrix $\mathbf{R}(\phi, \theta, \psi)$ being expressed as follows.

$$\mathbf{R}(\phi, \theta, \psi) = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{6.6}$$

Where the angles $\theta$ and $\phi$ can be found using the expression below.

$$\mathbf{n} = \begin{bmatrix} -\sin\theta \\ \sin\phi\cos\theta \\ \cos\phi\cos\theta \end{bmatrix} \tag{6.7}$$

It can be seen that there is no way to reconstruct the angle $\psi$ from the normal vector $\mathbf{n}$. This is a rather intuitive result, since vector $\mathbf{n}$ does not encode any heading information. Hence, $\psi$ needs to be provided to give a unique solution. It is noted that the knowledge of the angle $\psi$ does not really help to recover the shape of the trajectory, but merely indicates from which direction is the landing site being approached. As a result, this assumption is considered to not affect the contribution of the work.

In summary, the minimum required information to make the problem of velocity inference well-posed are:

- set of three or more optical flow measurements $\{\hat{\mathbf{u}}(\hat{x}, \hat{y}) \mid \hat{x}, \hat{y} \in \mathcal{P}_{mc}\}$ (four if $\boldsymbol{\omega_b}$ is not known)
- additional metric measurement, like e.g. an altitude reading, to resolve scale ambiguity in Equation 6.1
- yaw angle $\psi$ to determine the heading in the matrix $\mathbf{R}(\phi, \theta, \psi)$ shown in Equation 6.6

However, as outlined in the following Subsection 6.1.2, further information may be provided to increase the accuracy of the solutions.

Lastly, egomotion related error metrics need to be established in order to be optimized. As seen in Equation 6.1, a simple sum of the Egomotion Error ($EE$), as shown below, is insufficient on itself. This is because the translational component of optical flow scales indirectly with altitude, causing the metric to be dominated by high-altitude segments of the trajectory. In these regions, even a small optical flow error can result in a significant $EE$.

$$EE = \|\mathbf{v_{est}} - \mathbf{v_{gt}}\| \tag{6.8}$$

Hence, following altitude-insensitive Altitude Corrected Egomotion Error (ACEE) is introduced as follows to mitigate this issue.

$$ACEE = \frac{\|\mathbf{v_{est}} - \mathbf{v_{gt}}\|}{-z} \tag{6.9}$$

### 6.1.2 Information regimes

As already noted in Subsection 2.1.2 it may be beneficial to provide additional information above the minimum required set to improve the estimates. It is important to study the effect of the exact information sets on accuracy, especially with regards to solving the aforementioned scale ambiguity in the Equation 6.1.

In the current time, lightweight Sagnac effect gyroscopes are fairly widespread. These instruments, as pointed out by Armenise et al. (2011), reach much higher precisions when estimating rotational rates than would be expected from an optical navigation system such as the one treated in this work. Hence, for the purpose of realism, we will consider the rotational rate $\boldsymbol{\omega_b}$ to be known from a hypothetical gyroscope for all our experiments.

Furthermore, three regimes are proposed to evaluate the sensitivity of the results to known information. They are named as and listed with increasing difficulty as follows.

- Known Altitude and Attitude Regime (KAA)
- Rangefinder Regime (R)
- Known Altitude Regime (KA)

Known Altitude and Attitude regime is included to investigate possible precision of estimates in a case, where the spacecraft is well instrumented and the altitude and attitude readings are provided to estimate the velocity $\mathbf{v}$. It is noted that as can be seen from Section 6.2, it does not matter whether the known variable is altitude or another range reading such as the one described in the next paragraph. In reality, this regime could easily correspond to a lander, which is e.g. utilizing a gyroscope, star tracker and a rangefinder. This regime is considered to be closest to probable use of event camera.

The Rangefinder Mode is meant to represent a realistic scenario, where a lander is flying with the true minimum set of instruments required plus a gyroscope. A forward-pointing rangefinder is providing the scale information to resolve the scale ambiguity through a reading $l_{rf}$ as shown in the Figure 6.1. The attitude is considered unknown. It may seem strange to include a scenario featuring both unknown attitude and a gyroscope at the same time, but independent attitude measurements absolve the spacecraft from requiring any precise initial attitude estimate or a low gyroscope drift. Lastly, the results are strongly indicative of potential with regards to landing site slope reconstruction, which is another important problem from similar class.

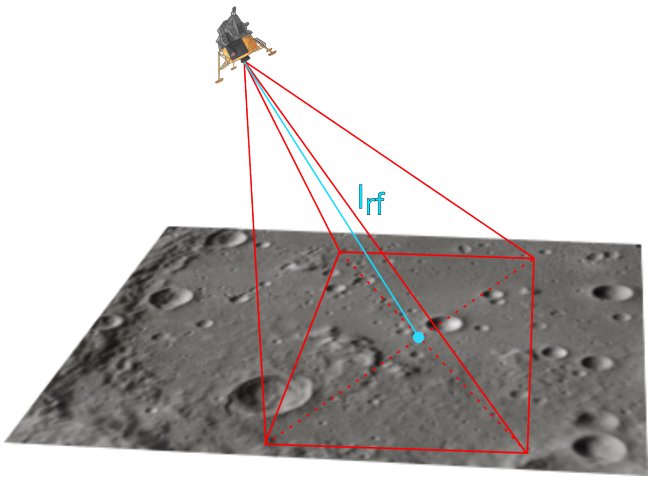



Figure 6.1: The event camera field of view (in red) and the rangefinder (in blue). The meaning of distance $l_{rf}$ can be seen from the figure.

Then comes the hardest Known Altitude regime. Here the actual altitude above surface is provided directly by e.g. radar altimeter. From the perspective of landing on planetary bodies with an atmosphere, it is even more practical as it could e.g. utilize information from pressure altimeter. As this is a common problem in the area of unmanned aerial vehicle control, it also provides direct tie to other works in this field like Grabe et al. (2015) or P. Li et al. (2015).

Table 6.1 gives an overview of the additional known variables in each of the regimes.

| **Regime** | **z** | $\phi$ | $\theta$ | $\psi$ | **p** | **q** | **r** | **Rangefinder** |
|---|---|---|---|---|---|---|---|---|
| KAA | • | • | • | • | • | • | • | |
| KA | • | | | • | • | • | • | |
| R | | | | • | • | • | • | • |

Table 6.1: Known additional variables for each regime.

## 6.2 Solving the overdetermined system for egomotion

This section will discuss the least-squares solutions to an overdetermined system reached by stacking multiple instances of Equation 6.1 from different point inferences. As it will be seen, this solution greatly differs when additional information are introduced. First, the KAA regime will be discussed followed by the R regime and the KA regime.

### 6.2.1 Known Altitude and Attitude regime

In the case of the KAA regime, the solution is very simple and will be outlined in the next few paragraphs. As P. Li et al. (2015) notes and as can be seen from Equation 3.1, it is possible to rewrite Equation 6.4 for a pinhole camera with focal length $f$ in the following way.

$$\frac{n_X(X/Z) + n_Y(Y/Z) + n_Z}{-z} = \frac{n_X(\hat{x}/f) + n_Y(\hat{y}/f) + n_Z}{-z} = \frac{1}{Z} \tag{6.10}$$

Equation 6.1 can then be rewritten in a way shown in Equation 6.11.

$$\hat{\mathbf{u}}(\hat{x},\hat{y}) - \mathbf{K}(\hat{x},\hat{y})\boldsymbol{\omega_b} = \hat{\mathbf{u}}'(\hat{x},\hat{y}) = \frac{n_X(\hat{x}/f) + n_Y(\hat{y}/f) + n_Z}{-z} \begin{bmatrix} f & 0 & -\hat{x} \\ 0 & f & -\hat{y} \end{bmatrix} \begin{bmatrix} v_{b,X} \\ v_{b,Y} \\ v_{b,Z} \end{bmatrix} \tag{6.11}$$

The vector $\mathbf{n}$ can be found from angles $\phi$ and $\theta$ using Equation 6.7 and the altitude $z$ is known. Hence, the Equation 6.11 can be stacked from different points to create a system of a form shown by 6.12. This system then can be solved for least-squares fit easily as matrix $\mathbf{A}$ is full rank.

$$\frac{1}{-z}\mathbf{A}\mathbf{h} = \mathbf{b} \tag{6.12}$$

### 6.2.2 Rangefinder regime

The R regime using a rangefinder is much more complex than the KAA regime. This is due to the attitude, and by extension vector $\mathbf{n}$, being considered unknown. First, it is noted that the as can be seen from Figure 6.2, the rangefinder reading $l_{rf}$ relates to altitude $z$ as follows.

$$-z = n_Z l_{rf} \tag{6.13}$$

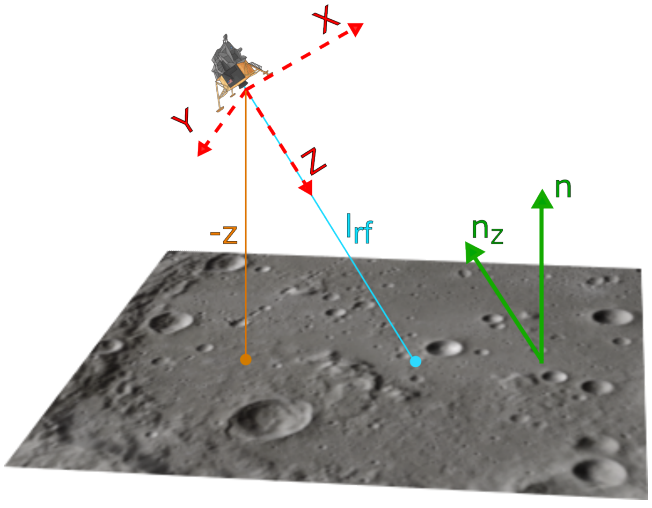



Figure 6.2: Normal vector $\mathbf{n}$ (green), reference frame $\mathcal{F}_b$ (red) and the relevant distances $l_{fr}$ (blue) and $-z$ (orange). Note the two similar triangles, which lead to $n_Z/1 = -z/l_{rf}$

Similarly to previous section, we can combine Equations 6.10, 6.4 and this time also Equation 6.13 to reach Equation 6.14.

$$\frac{(n_X/n_Z)(\hat{x}/f) + (n_Y/n_Z)(\hat{y}/f) + 1}{l_{rf}} = \frac{1}{Z} \tag{6.14}$$

Combining equations 6.1 and 6.14 leads to a following expression where $\hat{\mathbf{u}}'(\hat{x},\hat{y})$ is a de-rotated optical flow.

$$\hat{\mathbf{u}}(\hat{x},\hat{y}) - \mathbf{K}(\hat{x},\hat{y})\boldsymbol{\omega_b} = \hat{\mathbf{u}}'(\hat{x},\hat{y}) = \frac{1}{l_{rf}} \begin{bmatrix} \hat{x}\hat{y} & \hat{y}^2 & 1 & 0 & 0 & 0 & -\hat{x}^2 & -\hat{x}\hat{y} & -\hat{x} \\ 0 & 0 & 0 & \hat{x}\hat{y} & \hat{y}^2 & 1 & -\hat{x}\hat{y} & -\hat{y}^2 & -\hat{y} \end{bmatrix} \begin{bmatrix} (n_X/n_Z)v_{b,X} \\ (n_Y/n_Z)v_{b,X} \\ v_{b,X} \\ (n_X/n_Z)v_{b,Y} \\ (n_Y/n_Z)v_{b,Y} \\ v_{b,Y} \\ (n_X/n_Z)v_{b,Z} \\ (n_Y/n_Z)v_{b,Z} \\ v_{b,Z} \end{bmatrix} \tag{6.15}$$

Equation 6.16 is generated by stacking Equation 6.15 from various points. As can be seen from Equation 6.15 matrix $\mathbf{A}$ is rank deficient with rank five in this case, so a more elaborate solution process in comparison to KAA regime is required. Moreover, many of the rows will be almost linearly dependent as they describe optical flow vectors in locations very close to each other on the image plane, which can potentially lead to numerical instabilities.

$$\frac{1}{l_{rf}}\mathbf{A}\mathbf{h} = \mathbf{b} \tag{6.16}$$

One of the proven methods to solve systems like Equation 6.16 is the so-called 2-SVD method outlined by P. Li et al. (2015), which uses the singular value decomposition to handle the rank deficiency and other issues. The rest of this subsection will describe the use 2-SVD method based on the work P. Li et al. (2015).

First the left hand side of the system described by Equation 6.16 is multiplied by its own transpose and factorized using SVD as described by the Equation 6.17.

$$\mathbf{A^T A} = \mathbf{U_1 D_1 V^T_{m,1}} \tag{6.17}$$

The right hand side can be expressed in terms of the matrices obtained from this SVD, with $k_{svd}$ being a scaling factor.

$$\frac{1}{l_{rf}}\mathbf{h} = \mathbf{V_{m,1}e} + k_{svd}\mathbf{v_m}, \tag{6.18}$$

where $V_{m,1}$ is the $9 \times 9$ orthogonal matrix, and $\mathbf{v_m}$ represents its last column:

$$\mathbf{v_m} = \begin{bmatrix} \mathbf{V_{m,1},(1,9)} \\ \mathbf{V_{m,1},(2,9)} \\ \vdots \\ \mathbf{V_{m,1},(9,9)} \end{bmatrix}.$$

The vector $\mathbf{e}$ is computed as:

$$\mathbf{e}_i = \frac{\mathbf{b'}_i}{\mathbf{d}_i}, \quad i \in \{1, 2, \ldots, 9\},$$

$$\mathbf{b'} = \mathbf{U_1^T A^T b}, \quad \mathbf{d}_i = \mathbf{D_{1,ii}}$$

Then P. Li et al. (2015) notes that the vector $\mathbf{h}$ can be reshaped into a matrix $\mathbf{H}$ in the following way using function $M$.

$$\frac{1}{l_{rf}}\mathbf{H} = \frac{1}{l_{rf}}M(\mathbf{h}) = \frac{1}{l_{rf}} \begin{bmatrix} (n_X/n_Z)v_{b,X} & (n_Y/n_Z)v_{b,X} & v_{b,X} \\ (n_X/n_Z)v_{b,Y} & (n_Y/n_Z)v_{b,Y} & v_{b,Y} \\ (n_X/n_Z)v_{b,Z} & (n_Y/n_Z)v_{b,Z} & v_{b,Z} \end{bmatrix} = M(\mathbf{V_{1,m}e}) + k_{svd}M(\mathbf{v_m}) \tag{6.19}$$

The scaling factor $k_{svd}$ is still unknown. But as it can be easily observed from the Equation 6.19, the matrix $\mathbf{H}$ is a result of an outer product of vectors $\mathbf{n}/n_z$ and $\mathbf{v_b}$. Hence, the rank of the matrix $\mathbf{H}$ must be one and its determinant must be equal to zero and so should the determinants of all its 2 x 2 submatrices. To utilize this property, P. Li et al. (2015) proposes to solve the characteristic equation of matrix $\mathbf{H}$, which is a third-degree polynomial, and then pick the root which minimizes the sum of the submatrix determinants. As seen in Equation 6.19, if matrix $\mathbf{H}$ is found, obtaining $\mathbf{v_b}$ is trivial and can be done by multiplying third column of the matrix by the range $l_{rf}$.

Lastly, the vector $\mathbf{n}$ can be found as shown below using another SVD.

$$\mathbf{H} = \mathbf{U_2 D_2 V_{m,2}^T} \tag{6.20}$$

$$\mathbf{n} = \begin{bmatrix} \mathbf{V_{m,2},(1,1)} \\ \mathbf{V_{m,2},(2,1)} \\ \mathbf{V_{m,2},(3,1)} \end{bmatrix}.$$

It is noted that the normal vector $\mathbf{n}$ still has two solutions, so $n_Z > 0$ is enforced. The angles $\phi$ and $\theta$ then simply stem from Equation 6.7. Hence, $\mathbf{v}$ can be found from Equation 6.6.

$$\theta = -arcsin(n_X) \tag{6.21}$$

$$\phi = arcsin\left(\frac{n_Y}{cos(\theta)}\right) \tag{6.22}$$

### 6.2.3 Known Altitude regime

The solution for KA regime with known altitude significantly resembles the R regime with the known rangefinder reading. Again, the main issue is to find the normal vector $\mathbf{n}$ due to unknown attitude. However, the solution is more difficult than in R regime, because instead of the slanted range to the surface location where the spacecraft points, it is necessary to operate just with altitude.

Combining Equations 6.1 and 6.10 leads to a following expression where $\hat{\mathbf{u}}'(\hat{x}, \hat{y})$ is a de-rotated optical flow.

$$\hat{\mathbf{u}}(\hat{x}, \hat{y}) - \mathbf{K}(\hat{x}, \hat{y})\boldsymbol{\omega_b} = \hat{\mathbf{u}}'(\hat{x}, \hat{y}) = \frac{1}{-z}\begin{bmatrix} \hat{x}\hat{y} & \hat{y}^2 & 1 & 0 & 0 & 0 & -\hat{x}^2 & -\hat{x}\hat{y} & -\hat{x} \\ 0 & 0 & 0 & \hat{x}\hat{y} & \hat{y}^2 & 1 & -\hat{x}\hat{y} & -\hat{y}^2 & -\hat{y} \end{bmatrix} \begin{bmatrix} n_X v_{b,X} \\ n_Y v_{b,X} \\ n_Z v_{b,X} \\ n_X v_{b,Y} \\ n_Y v_{b,Y} \\ n_Z v_{b,Y} \\ n_X v_{b,Z} \\ n_Y v_{b,Z} \\ n_Z v_{b,Z} \end{bmatrix} \tag{6.23}$$

An overdetermined system described in the Equation 6.24 is reached by stacking the Equation 6.23 from more than three points.

$$\frac{1}{-z}\mathbf{A}\mathbf{h} = \mathbf{b} \tag{6.24}$$

Then the 2-SVD method described in Subsection 6.2.2 is used to solve this system for the homography matrix $\mathbf{H}$ shown below and the vector $\mathbf{n}$.

$$\mathbf{H} = \begin{bmatrix} n_X v_{b,X} & n_Y v_{b,X} & n_Z v_{b,X} \\ n_X v_{b,Y} & n_Y v_{b,Y} & n_Z v_{b,Y} \\ n_X v_{b,Z} & n_Y v_{b,Z} & n_Z v_{b,Z} \end{bmatrix} \tag{6.25}$$

With the homography matrix $\mathbf{H}$ known, the following relation is clearly valid as $\mathbf{n}$ is a unit vector. This yields the desired velocity $\mathbf{v_b}$.

$$\mathbf{H}\mathbf{n} = \frac{1}{-z}\mathbf{v_b} \tag{6.26}$$

## 6.3 Error propagation and sensitivity to spatially correlated errors

This section will explore how errors in optical flow estimation propagate into errors in egomotion estimation using Monte Carlo method experiments. First, simple case of uncorrelated optical flow errors will be dealt with. Then the section will dive into the more complex and malign scenario, where the optical flow errors are spatially correlated on the image plane.

### 6.3.1 Error propagation and sensitivity

In order to establish the sensitivity to errors in the estimated optical flow, the following Monte Carlo method experiment was conducted. Six optimal trajectories were generated with two of them belonging to the ventral type, two to the nominal type, and two of them representing the diverts with high angular velocities. Similarly to the real experiments described in Chapter 7, 200x200 px motion fields $\mathbf{U}_{\hat{x}}$, $\mathbf{U}_{\hat{y}}$, belonging to a camera with a 45 degree field of view, were calculated based on Equation 6.1 for each of the trajectories with a 10 Hz frequency. For each of these motion fields, the perturbation matrices $\mathbf{E}_{e,\hat{x}}$ and $\mathbf{E}_{e,\hat{y}}$ were generated as follows. First, a matrix of angular errors $\mathbf{E}_a$ is created randomly with an independent value for each pixel. Then a value of AEE is randomly drawn, which will represent the endpoint error in all the points of the field after the perturbation.

$$\mathbf{E}_a = \begin{bmatrix} \mathbf{E}_{\mathbf{a},11} & \mathbf{E}_{\mathbf{a},12} & \cdots & \mathbf{E}_{\mathbf{a},1W} \\ \mathbf{E}_{\mathbf{a},21} & \mathbf{E}_{\mathbf{a},22} & \cdots & \mathbf{E}_{\mathbf{a},2W} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{E}_{\mathbf{a},W1} & \mathbf{E}_{\mathbf{a},W2} & \cdots & \mathbf{E}_{\mathbf{a},WW} \end{bmatrix} \quad \text{where} \quad \mathbf{E}_{\mathbf{a},\hat{x}\hat{y}} \sim \text{Uniform}(0, 2\pi) \tag{6.27}$$

$$\mathbf{E}_{\mathbf{e},\hat{\mathbf{x}}} = \sin(\mathbf{E}_\mathbf{a})\,\text{AEE}, \tag{6.28}$$

$$\mathbf{E}_{\mathbf{e},\hat{\mathbf{y}}} = \cos(\mathbf{E}_\mathbf{a})\,\text{AEE}, \tag{6.29}$$

$$\text{where} \quad \text{AEE} \sim \text{Uniform}(0, 10). \tag{6.30}$$

Then the motion field is perturbed as shown below.

$$\tilde{\mathbf{U}}_{\hat{x}} = \mathbf{U}_{\hat{x}} + \mathbf{E}_{\mathbf{e},\hat{\mathbf{x}}} \tag{6.31}$$

$$\tilde{\mathbf{U}}_{\hat{y}} = \mathbf{U}_{\hat{y}} + \mathbf{E}_{\mathbf{e},\hat{\mathbf{y}}} \tag{6.32}$$

The 200×200 field is divided into 16 squares and from each of the squares the midpoint is selected and added to the set $\mathcal{P}_{mc}$, as illustrated in Figure 6.3. The perturbed optical flow at these points is then used to construct the set $\{\tilde{\mathbf{u}}(\hat{x}, \hat{y}) \mid \hat{x}, \hat{y} \in \mathcal{P}_{mc}\}$. The optical flows at points in this set are subsequently processed using the aforementioned algorithms to obtain the egomotion and the ACEE values.

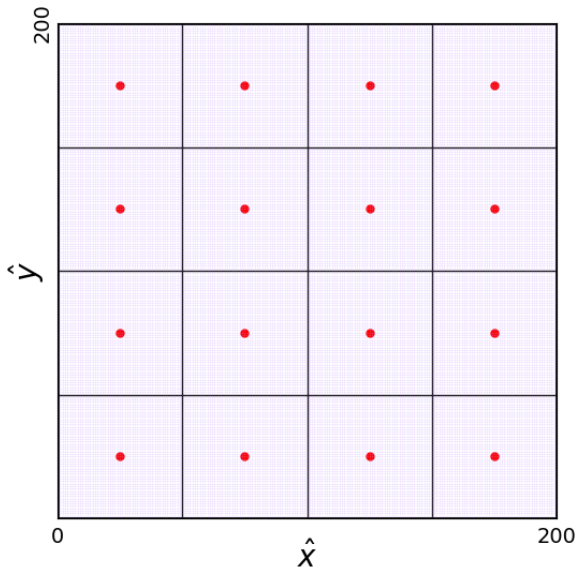



Figure 6.3: Example of point sampling from a 200x200 square. The square is divided into 16 subsquares and a midpoint is chosen from each.

The Figure 6.4 shows the resultant mean and standard deviation of ACEE as a function of AEE for all the three information regimes. It is notable that at higher values of AEE the curve behaves almost linearly and as such justifies the use of the simple AEE loss in Section 5.4.

Lastly, it is noted that if e.g. a requirement of mean absolute egomotion error of 10 m/s at the high gate was set, which equates to ACEE of 0.004 even in the simplest KAA regime, this leads to an AEE value well below 0.5. Such performance is well above the current state-of-the-art not even with spiking neural networks, but also most other methods. Hence, a recommendation is made to explore further possibilities like e.g. feature tracking to improve this performance. Another possibility is increasing the camera resolution.

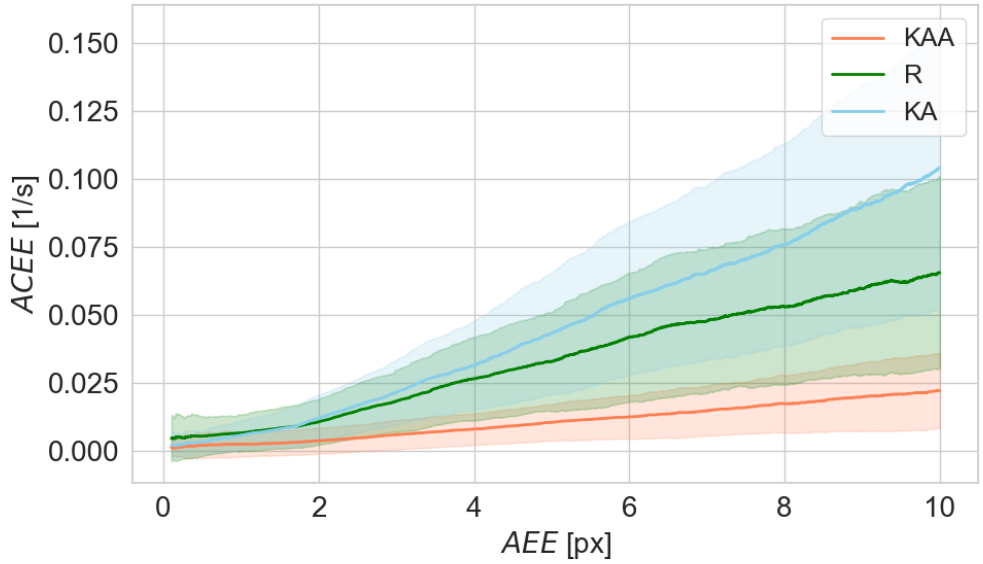



Figure 6.4: Mean and one standard deviation of ACEE as a function of AEE as a result of Monte Carlo experiments for the three information regimes.

### 6.3.2 Spatially correlated errors

Preliminary experiments with the neural networks revealed that they often tend to spatially smoothen the estimated flow. This leads to a major issue where the angular and, to a certain extent, even magnitude errors on individual pixels are not independent. As such, increasing the number of input points does not lead to increase in performance above a certain threshold. To study this issue and put a lower bound on its severity, the following scheme, which simulates the spatially correlated angular errors, was used.

Similarly to the previous subsection, a matrix of random angles $E_a$ is generated at each step of the Monte Carlo simulation in order to find the matrix $E_{e,x}$ and $E_{e,y}$ which represent the two components of the total error on each pixel. However, in this case the matrix $E_a$ is generated with dimensions smaller than the $E_{e,x}$ and $E_{e,y}$ by a certain factor of and then it is bilinearly upsampled by this respective factor. As such, the errors in the resultant matrices $E_{e,x}$ and $E_{e,y}$ vary smoothly and they suffer from a certain level of spatial correlation.

Figure 6.5 shows mean values of KAA regime ACEE, generated by a Monte Carlo simulation from 16 points, plotted against values of AEE with varying average level of correlation loss term AC across the run. As can be observed, higher AC can lead to a massive increase in ACEE for constant AEE. The exact results are omitted here, but in R and KA regimes, this effect is even more pronounced. Moreover, as previously noted, in a realistic scenario, the errors may be also spatially correlated with magnitude, and these effects may be even worse.

Of course, the number of point flow inferences does not have to be 16; this value was determined through experiments on real data as a threshold beyond which further increases do not improve performance. However, the main idea stays the same - to stabilize noise and reach good ACEE values, the point inferences should be as independent as possible.

## 6.4 Navigational filter

The motion estimates, despite often rather violently fluctuating, typically remain centered around a well-defined mean. As the interest is in the mean, rather than the noisiness, filtering becomes essential. However, this filtering is not the main scope of this work and it can possibly bring multiple additional tuning variables, if e.g. a Kalman

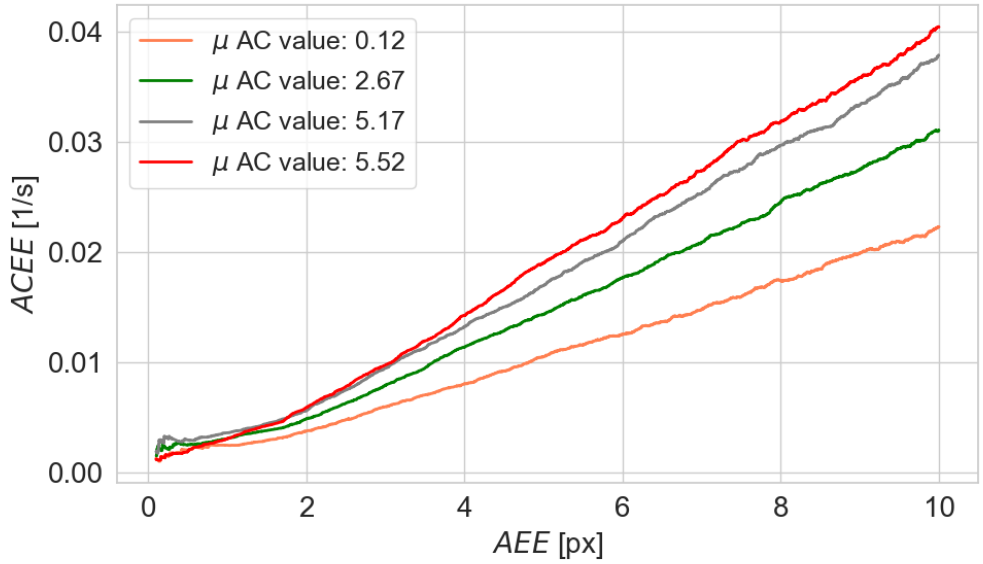



Figure 6.5: Adverse effects of smooth optical flow field with spatially correlated angular errors illustrated by Monte Carlo experiments. Each of the runs has a different average value of the AC loss term where increase signifies higher spatial correlation of errors.

filter is used. Hence, two simple discrete low-pass filters were designed and tuned for velocity $\mathbf{v}$ (for all the regimes) and normal vector $\mathbf{n}$ (for R and KA regimes). They are not claimed to be the best performing filters for the purpose, but they help to recognize the actual performance of the estimators rather than just measure their noisiness. Figure 6.6 shows how these two filters are utilized.

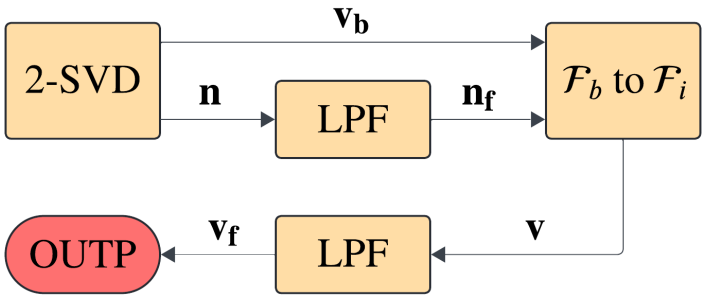



Figure 6.6: Setup of the two low-pass filters used in the R and KA regime.

The velocity filter can be simply expressed as follows with $\mathbf{v_f}$ being the filtered velocity vector.

$$\mathbf{v_{f,k+1}} = \beta_v \mathbf{v_k} + (1 - \beta_v)\mathbf{v_{k+1}} \tag{6.33}$$

With regards to normal vector $\mathbf{n}$ the situation is more complicated as some of the attitude changes are really fast. Hence a decision was made to correct for the rotations using the known angular velocities. The final filter takes the form shown below with $\mathbf{n}_f$ is the filtered value of the vector.

$$\mathbf{n}_{f,k+1} = \beta_n \left( \mathbf{n}_{f,k} + \frac{d\mathbf{n}}{dt} \Delta t \right) + (1 - \beta_n)\mathbf{n}_{k+1} \tag{6.34}$$

The derivative $d\mathbf{n}/dt$ can be expressed in the following fashion, while the expressions for $\partial\theta/\partial t$ and $\partial\phi/\partial t$ can be found in Equation 3.

$$\frac{d\mathbf{n}}{dt} = \frac{\partial \mathbf{n}}{\partial \theta}\frac{\partial \theta}{\partial t} + \frac{\partial \mathbf{n}}{\partial \phi}\frac{\partial \phi}{\partial t} \tag{6.35}$$

$$\frac{d\mathbf{n}}{dt} = \begin{bmatrix} -\cos\theta \\ -\sin\phi\sin\theta \\ -\cos\phi\sin\theta \end{bmatrix} \frac{\partial\theta}{\partial t} + \begin{bmatrix} 0 \\ \cos\phi\cos\theta \\ -\sin\phi\cos\theta \end{bmatrix} \frac{\partial\phi}{\partial t}. \tag{6.36}$$

Values of $\beta_v = \beta_n = 0.8$ were found to function well. Lastly, it is noted that experiments revealed that filtering the estimation outputs proved to be much more effective than trying to spatially or temporally filter the inputs. Experimentation was done with confidence "planarity" measures $c_p$ of the following type similar to suggestions of Grabe et al. (2015), but rejection of erroneous results using this method did not measurably improve performance.

$$c_p = \left\| \mathbf{b} + \frac{1}{z} A\mathbf{h} \right\| \tag{6.37}$$

# Chapter 7

# Results

This chapter contains the results of the egomotion estimation experiments. First, performance on the ACEE metric, introduced in Equation 6.9, will be elaborated upon in Section 7.1. Then it will be shown how large the errors are in comparison to the velocity at each altitude in Section 7.2.

## 7.1 Altitude-Corrected Egomotion Errors

This section will first discuss altitude independence of the ACEE metric in order to establish whether it can meaningfully describe an entire trajectory. Then it will discuss the measured ACEE values for both the OES AC and the BM models. It is noted that point inference sampling is identical to the sampling used in Section 6.3.

### 7.1.1 Altitude independence

To show that a navigation performance of the optical system on a trajectory can be summarized into the single ACEE parameter, it is necessary to show that ACEE does not vary throughout the trajectories. Figure 7.1 shows the ACEE plot against altitude for both the OES AC and the BM models. It can be seen that ACEE is approximately constant with altitude except a strong spike below altitude of cca. 300 meters. However, in this problematic region sharp attitude maneuvers to meet the final conditions occur and they can easily explain this degraded performance as discussed in Section 5.6.3.

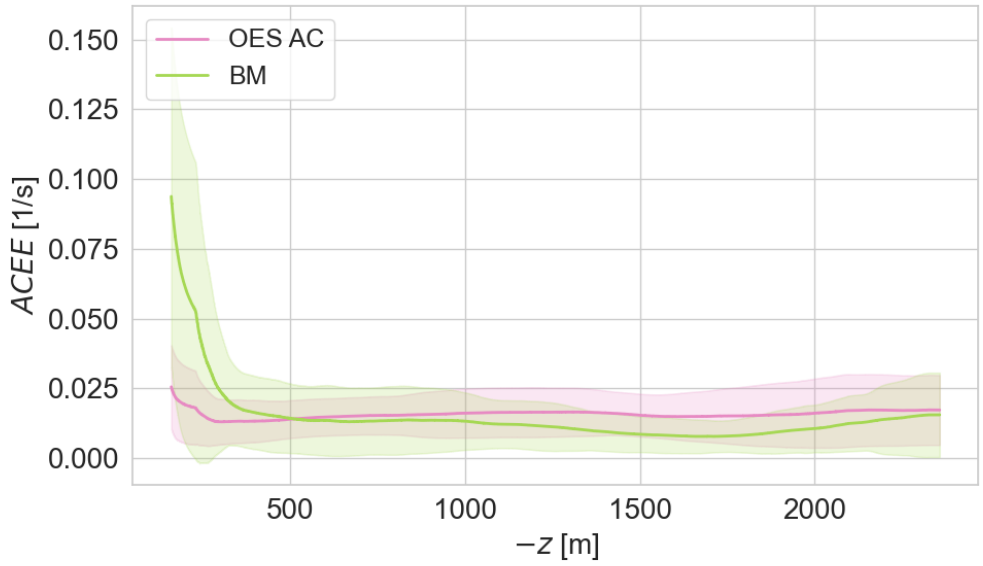



Figure 7.1: ACEE plotted against the altitude $z$ for OES AC and BM models. The spike under altitude of 300 m is easy to explain by fast attitude maneuvers before landing, which occur in this region and degrade the accuracy.

Hence, from now on, ACEE will be used as one single parameter to summarize precision of the navigation system on a trajectory. The reader is reminded that ACEE is defined in Equation 6.9 and can easily be converted to absolute egomotion errors through multiplication by altitude. If ACEE of 0.02 1/s is measured for a trajectory, then the mean error at 100 m altitude will be 2 m/s and at 1000 m it will be 20 m/s.

### 7.1.2 Performance in different regimes

This subsection will introduce the performance data for both of the OES AC and BM models and all the trajectory types.

#### OES AC

Figure 7.2 shows the plots of ACEE against altitude when using the OES AC model. There are multiple things to note.

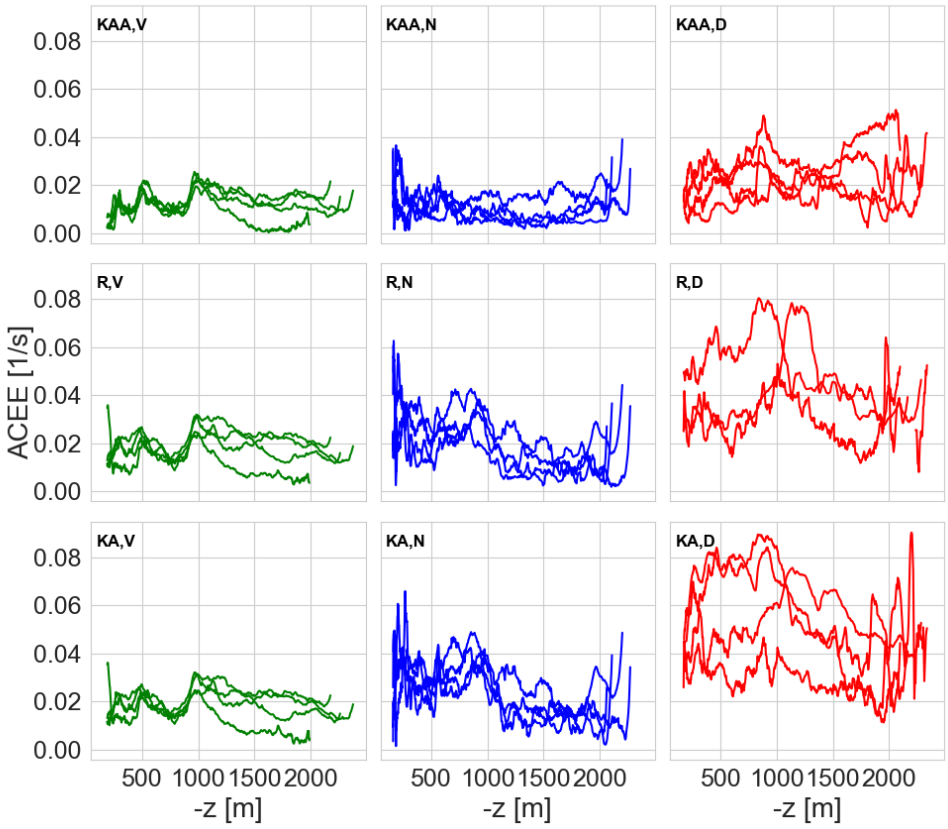



Figure 7.2: ACEE inferred using the OES AC model plotted against the altitude $z$ for all the test trajectories.

First of all, it can be seen that the navigational filter converges very quickly and as such does not affect the results much. If all of the datapoints above 2000 meters are ignored, it is possible to obtain values with marginal filter influence.

Secondly, the ventral trajectories are estimated rather poorly at higher altitudes, but quickly show better results in the last part of the trajectory. What is particularly interesting is that around altitude of 1000 meters, the errors for the four test trajectories converge together in a rather suspicious manner. As can be seen from Figure 2.3, all of the ventral trajectories will have the same vertical velocity and consequently optical flow in this part of the trajectory, because of the nature of the optimal ventral trajectory solution. Even though the input is varied as all of the trajectories have a slightly different landing site position, they still produce very similar error for the respective optical flow. It is stated that this could be evidence of overfitting to certain flow field structures.

Thirdly, the nominal trajectories show a very strong degradation in performance in the last cca. 300 meters above the low gate where attitude maneuvers occur. As already discussed in Subsection 7.1.1, this should come

as no surprise as the fast attitude maneuvers truly degrade the performance.

Table 7.1 offers means and standard deviations for ACEE values in the trajectories. However, to give a more meaningful estimate, all datapoints above 2000 meters and below 300 meters of altitude are omitted.

| **Regime/Trajectory type** | **V** | **N** | **D** |
|---|---|---|---|
| **KAA** | 0.012 ($\sigma = 0.005$) | 0.010 ($\sigma = 0.005$) | 0.023 ($\sigma = 0.009$) |
| **R** | 0.017 ($\sigma = 0.007$) | 0.019 ($\sigma = 0.009$) | 0.040 ($\sigma = 0.016$) |
| **KA** | 0.017 ($\sigma = 0.007$) | 0.024 ($\sigma = 0.010$) | 0.048 ($\sigma = 0.020$) |

Table 7.1: Mean values of ACEE and their standard deviations when using the OES AC model. Datapoints above 2000 meters and below 300 meters are not included.

### BM

The BM model is mostly similar to OES AC in behavior, but there are a few differences which will be introduced in the following paragraphs.

It can be seen that BM performs well in the KAA regime and with ventral and nominal trajectories while outside of the final attitude maneuvers. However, the combination of R or KA regime with the diverts or the final parts nominal trajectories brings extremely poor results. This sudden degradation can be explained by failure to correctly estimate the normal vector **n**. A generally valid explanation for this behavior has not yet been found. It is of note, that the convergence of estimates on the ventral trajectories below 1000 meters, so well visible with the OES AC model, is not visible with the BM model.

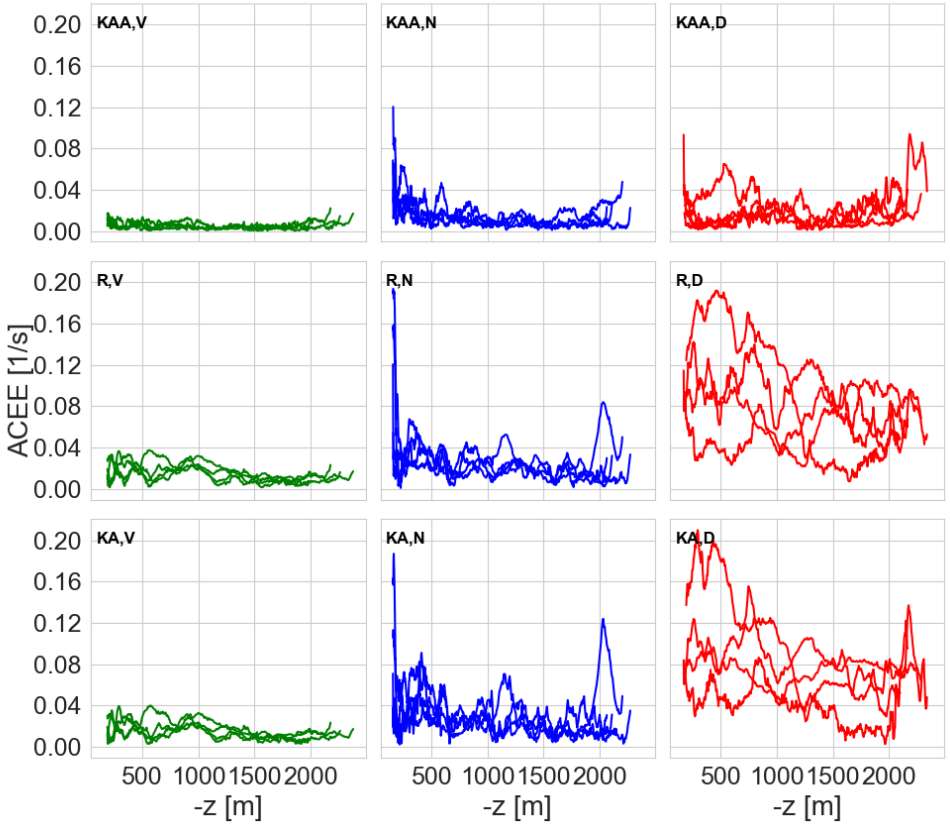



Figure 7.3: ACEE inferred using the BM model plotted against the altitude $z$ for all the test trajectories.

Table 7.2 offers the ACEE values tied to the BM model and their standard deviations for each of the trajectory types and information regimes.

| Regime/Trajectory type | V | N | D |
|---|---|---|---|
| **KAA** | 0.005 ($\sigma = 0.003$) | 0.012 ($\sigma = 0.007$) | 0.016 ($\sigma = 0.012$) |
| **R** | 0.015 ($\sigma = 0.008$) | 0.023 ($\sigma = 0.012$) | 0.076 ($\sigma = 0.042$) |
| **KA** | 0.015 ($\sigma = 0.008$) | 0.028 ($\sigma = 0.017$) | 0.072 ($\sigma = 0.037$) |

Table 7.2: Mean values of ACEE and their standard deviations when using the BM model. Datapoints above 2000 meters and below 300 meters are not included.

Table 7.3 offers a comparison between the OES AC and BM models. It can be observed that OES AC is dominant compared to BM on complex trajectories and when the attitude is unknown. On the other hand, BM is superior on simple trajectories and when the attitude is known.

| Regime/Trajectory type | V | N | D |
|---|---|---|---|
| **KAA** | +140% | -17% | +44% |
| **R** | +13% | -17% | -47% |
| **KA** | +13% | -14% | -33% |

Table 7.3: ACEE difference between OES AC and BM models in percent. Higher values indicate poorer performance of the OES AC model relative to the BM model and vice versa.

## 7.2 Relative egomotion error

While the ACEE metric provides an excellent way to summarize the absolute values egomotion error on a trajectory with one number, it fails to tell how big these errors are in comparison with the actual velocity. To give some illustration, a plot of the average altitude-corrected velocity ($\|\mathbf{v}\|/z$ against altitude is provided in Figure 7.4 based on the test trajectories.

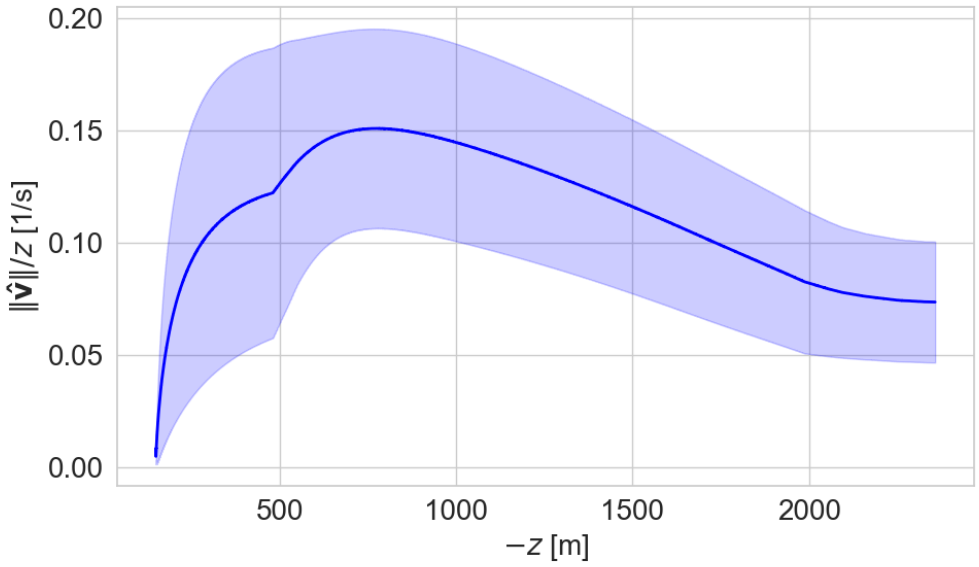



Figure 7.4: Altitude-corrected velocity against altitude in the test dataset. The line indicates mean and shaded areas indicate one standard deviation.

From the figure it can be seen that the relative errors will be the lowest around the altitude of cca. 800 meters and the highest at close to the high gate and at the low gate. It can also be seen that the egomotion errors will be mostly cca. one order of magnitude lower than the actual velocity with both models in the KAA regime. On the other hand, in case of diverts and KA regime, the error will be almost as large as the velocity itself at some points.

# Part III

# Final remarks

# Chapter 8

# Conclusion

In this chapter, answers to the research questions will be presented in Section 8.1. Then, recommendations for future research will be given in Section 8.2.

## 8.1 Answers to research questions

This section will present the answers to each of the research questions starting with the main research question.

> **RQ** - How accurately can vehicle egomotion be reconstructed during a fuel-optimal planetary landing using an event camera output processed with a spiking neural network?

A synthetic experimental set-up was created to test the performance of this navigation system in scenarios similar to the Apollo landing, and a network was trained. As there is a need for another sensor in such a case for absolute position-related values to resolve scale ambiguity, it was considered that data from a perfect forward-looking rangefinder are available. Moreover, a gyroscope providing angular velocities was assumed to be present. It was found that in such a scenario, the absolute error in the egomotion estimate will linearly increase with the altitude. In case of a simple ventral trajectory, the mean error and its standard deviation will obey the following formula.

$$\text{Egomotion Error} = \text{Altitude} \cdot (0.017 \pm \sigma = 0.007)[m/s] \tag{8.1}$$

Then in case of a trajectory representative of a nominal landing it will adhere to the formula below.

$$\text{Egomotion Error} = \text{Altitude} \cdot (0.019 \pm \sigma = 0.009)[m/s] \tag{8.2}$$

And on a divert trajectory including sharp attitude maneuvers it degrades to the following expression.

$$\text{Egomotion Error} = \text{Altitude} \cdot (0.040 \pm \sigma = 0.016)[m/s] \tag{8.3}$$

The validity of these results was tested between a high gate at 2286 m (7500 feet) and a low gate altitude at 152 m (500 feet).

It was found that these results can be sharply improved by providing additional attitude estimates and that on the other hand the results degrade when only altitude is provided instead of a forward-looking rangefinder.

For comparison, a near state-of-the-art hybrid spiking-analog network based on the work was trained. It was shown that this network outperforms the pure spiking network significantly on simple trajectories, but fails in comparison on complex trajectories with attitude maneuvers.

> **RSQ-1** - How can a synthetic event-based dataset for planetary landing be created?
>
> - **RSQ-1.1** - What is a suitable event camera simulator in this application?
> - **RSQ-1.2** - To what extent can such a dataset be validated?
> - **RSQ-1.3** - How can the appropriate trajectories for the purpose be generated?

Based on an existing design, a pipeline was devised which includes generating trajectories using combination of AMPL parser and SNOPT optimizer followed by natural scenery generation using PANGU software and conversion to events with v2e simulator. This combination proved adequate for the purpose and the choices were found to be justified.

An investigation was conducted, which compared the current state-of-the-art event camera simulators in terms of timestamp fidelity, event count and distribution fidelity, computational resources required, ease-of-use and flexibility. Despite its shortcomings in the timestamp fidelity area, the v2e simulator is the best as it excels in other fields.

A combined validation and tuning method is successfully devised, which includes data from Synthetic Lunar Terrain dataset. Focus is on a realistic interaction between the event camera pixels and edges on the surface in terms of event rate and its variance.

It is found that while the high-level camera and surface parameters like biases or contrast can be tuned, it is too difficult to extract any meaningful noise estimates from the ground truth event stream and readily available noise models must be relied upon.

Lastly, an analysis is performed to show that it is required to use general non-linear programming to calculate the optimal trajectories. Convex optimization is a simpler and readily available method for this purpose, but it fails to accurately estimate attitude maneuvers, which have a major impact on the optical flow and as a consequence on optical navigation.

**RSQ-2** - What is a good architecture of spiking neural network for processing the event stream during planetary landing?

- **RSQ-2.1** - How can the event timing information be well utilized in the spiking neural network?
- **RSQ-2.2** - What are the appropriate loss functions in the context of supervised learning of event-based optical flow?

Two U-Net like models, denoted as OES and ASN, focused on optical flow prediction were trained and evaluated. They were found to be competitive with a near state-of-the-art hybrid spiking-analog network. But, as can be seen from the answer to the main research question, their performance leaves much to be desired with the egomotion estimation errors sharply growing with altitude. This leaves open space for further investigation.

With regard to the first subquestion, the exotic OES network, which utilizes explicit temporal convolution and simple Integrate-and-Fire neurons to handle temporal information, confirmed to be superior to the more common ASN network which to a large degree relies on the inherent qualities of Leaky Integrate-and-Fire neurons. Considering that both options are considered neuromorphic hardware friendly in principle, it suggests that the method used in OES is comparatively better option moving forward.

An investigation was conducted using Monte Carlo method experiments to show that there is a major negative relationship between spatial correlation of the optical flow estimation errors and the resultant egomotion estimation errors. Moreover, it is observed that spiking neural networks have a tendency to smoothen the inferred flow and create such correlated fields. Based on this, traditional loss functions like average endpoint error are extended by a term punishing such behavior, leading to improved estimates.

**RSQ-3** - How can the egomotion be reconstructed from the output of a spiking neural network?

- **RSQ-3.1** - How should filtering be employed in order to increase performance?
- **RSQ-3.2** - In what conditions can useful egomotion estimates be generated from the optical flow?
- **RSQ-3.3** - How do the additional provided navigational information improve the egomotion estimates?

It is recognized that the egomotion estimation problem outlined by the main research question is a variant of a problem commonly found in drone navigation and appropriate solution is derived.

With regard to filtering of the flow, it is acknowledged that networks trained on temporally and spatially smooth motion fields associated with planetary landing tend to produce spatially and temporally smooth flow as well. This greatly limits effects of spatio-temporal filtering on the final results. Confidence measures based on planarity constraints are also found to have very marginal effect. On the other hand, a simple navigation low-pass filter keeping the egomotion and attitude estimates can help a lot to improve performance.

As mentioned previously, it is found that the egomotion estimation error increases linearly with altitude. Hence, at high altitudes, this method is comparatively less useful. Moreover, it is found that the optical flow error increases with the higher magnitude of optical flow. As such, e.g. sharp attitude maneuvers have a negative impact on the estimates. It is possible to observe that ventral descents are easiest to process, followed by nominal descents and diverts including maneuvering are the most difficult. As such, average errors twice as large compared to ventral trajectories are reached on diverts.

An estimation is also tested for a situation where attitude is known. It is found that the results are dramatically improved by cca. 30 to 50 percent depending on the trajectory nature. On the other hand, if the perfect rangefinder is replaced with a perfect altimeter, this leads to degradation of performance by cca. 0 to 20 percent.

## 8.2 Recommendations for future work

This section will briefly introduce recommendations for future work based on the experience obtained during this research.

As previously described, it was found to be very difficult to train dense optical spiking neural networks while retaining focus on low-level detail instead of global context. It is not a surprising result as the optical flow field during lunar landing is of very simple structure. However, this unfortunate effect could be possible to overcome by focus on tracking individual objects, e.g. craters, rather than estimating the dense optical flow. This line of research would be worth exploring. Other less complicated options include radical augmentation and masking, even though expected gains are questionable as the core of the problem is not truly addressed in this case.

Another possibility would be to use a dataset with ground truth optical flow fields which have less rigid structure. Although such fields would not occur naturally during landing, it is easy to obtain such dataset by rendering a certain amount of lunar scenes and then warping these images with randomized smooth optical flow. This should stimulate focus on the local context in learning.

If there were other attempts on learning the dense optical flow, it would be very interesting to incorporate one of the novel models, which came too late to influence this research. Above all, the spatiotemporal swin spikeformer of Tian and Andrade-Cetto (2024) performs very well on similar datasets and seemed to produce good results during informal tests by the author.

Lastly, the effect of the event camera resolution on the results was not studied in this work. However, as the lunar surface happens to be self-similar at the relevant scales, a higher resolution camera would see more detail and would possibly lead to better results. Variation of focal length is also of interest.

# Appendices

**Appendices A, B, C, and D are a part of separately graded literature study**

# Appendix A

# Event camera and event-based datasets

This chapter will introduce the bio-inspired vision sensors known as event cameras or silicon retinas. Firstly the Section A.1 will introduce the relevant functions in a biological eye, so that the reader can appreciate the biomimetic approach to vision taken by the event camera. Secondly the Section A.2 will introduce the basic event camera principles and their difference from frame-based cameras. Thirdly the Section A.3 will review the common available event cameras. Then, possible strategies which can be undertaken in order to obtain an event-based dataset are listed in Section A.4. A brief introduction of event camera simulators applicable on frame-based datasets will follow in Section A.5.

## A.1 Biological retinas

Even rather primitive vertebrates are able to solve complex navigational tasks. Perhaps the most incredible example are the little hummingbirds flying with an inch-perfect precision in order to drink nectar from flowers. A key enabler of this capability is the superb sight which allows them to obtain information during all kinds of motions and lighting conditions without prohibitive latency or energy consumption. This section is not meant to be a complete overview of the relevant vision systems and instead will only focus on a brief overview of vertebrate retinas which are the most relevant to the neuromorphic sensors described later. The section relies on Henley (2021) for the description.

### A.1.1 Structure of biological retinas

Vertebrate retinas consist of multiple layers of nerve cells which have different functions. For the purpose of this review only photoreceptor cells, bipolar cells and ganglion cells will be treated. However it is noted that there are also other types of cells, namely the amacrine and horizontal cells, which are however not very relevant to event cameras.

As shown in the Figure A.1 the first layer is formed by the photoreceptor cells which convert the light into nerve signals. The photoreceptor cells are connected to bipolar cells which react to light increments and decrements. There are so-called ON and OFF bipolar cells. The ON bipolar cells are excited by light increments and inhibited by light decrements while the OFF bipolar cells function in the exactly opposite manner.

The signal from bipolar cells serves as an input to ganglion cells. These cells act as an interface between the retina and the brain, They convert the signal from the ON and OFF paths into spikes which are compatible with the receiver.

### A.1.2 Features of biological retinas

The main takeaway from this short description is that human eye does register changes rather than still frames. This is very beneficial because the regions where the light significantly varies can contain edges, features and outlines which generally reveal much more information than uniform surfaces. They can for example help to recognize what object are we looking at or what is our own motion.

The resulting spikes are also sparse as a result of the signal being tied to change in brightness rather than any absolute value. This is very efficient because there is no need to process again static scenes which were already processed.

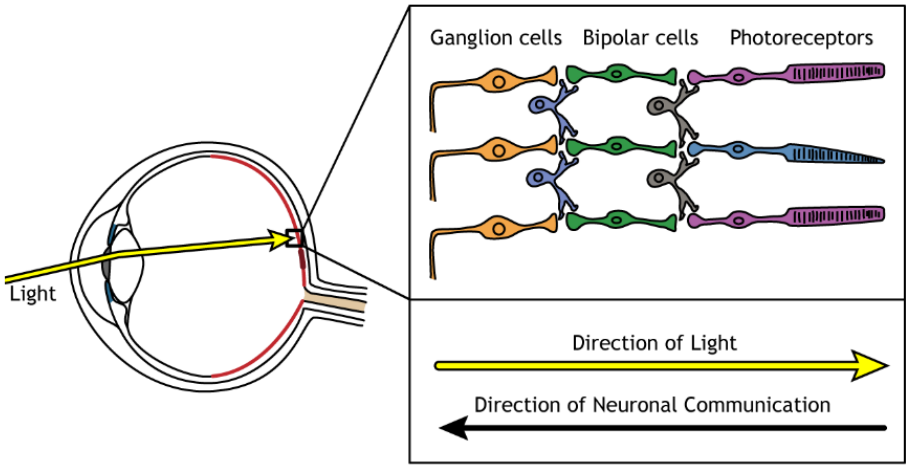



Figure A.1: Structure of a biological retina with an indicated direction of neural communication (image source: Henley (2021)).

## A.2 Event camera principles and advantages

The most common type of vision sensors are frame-based cameras. They became part of the daily life and they are found in large range of applications from smartphones to scientific instruments. Event cameras fundamentally differ from the frame-based cameras. This section will first discuss differences between the two imaging paradigms and compare their qualities. Secondly, it will discuss in closer detail how the event camera works on both functional and hardware sides.

### A.2.1 Differences between an event camera and a frame-based camera

The key difference between event camera and a frame-based camera is that the event camera asynchronously captures per-pixel change in brightness as stated by Gallego et al. (2019). This is in principle rather similar to a biological eye. On the other hand a frame-based camera captures an actual intensity with a predefined frequency. An illustration of this mechanism can be seen in the Figure A.2.

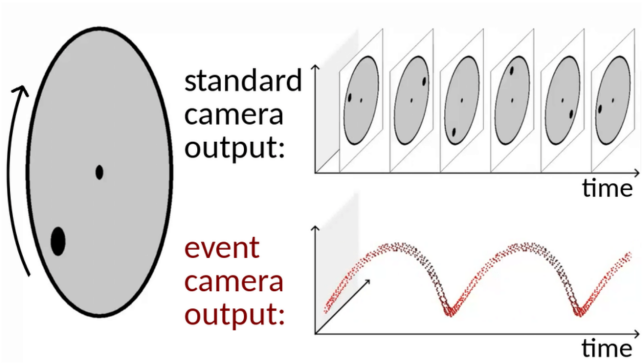



Figure A.2: Comparison of event camera and a frame-based camera outputs. The event camera produces an asynchronous stream of events while the frame based camera gathers the information into discrete time frames. Image taken from D. Gehrig et al. (2018).

This difference leads to a number of significant advantages and disadvantages between use of event and frame-based cameras. The main reasons to use an event camera as stated in Gallego et al. (2019) are their:

- High dynamic range
- Low latency
- High energy efficiency
- Resistance to motion blur

During a frame-based imaging process a shutter is open for a predetermined time which lets the light to the imaging sensor. This naturally limits the latency as information can be obtained only after the end of the period for which the shutter was open. As Kim et al. (2019) explains, it also limits the dynamic range (ratio between brightest and darkest part of the image) as individual pixels have limited well capacity. This means that if the shutter is open for too long, the pixels will become saturated while on the other hand too short opening will result in a noise dominated image. As Potmesil and Chakravarty (1983) state the shutter principle is also responsible for motion blur which will appear when the scene makes a significant movement while the shutter is open. Event camera lacks the shutter and as such does not suffer from the related issues. The energy efficiency comes with its sparse output which helps to avoid reprocessing completely static scenes repeatedly.

Such characteristics make the event camera extremely interesting for space applications. The often extreme lighting conditions, high velocities resulting in possibly prohibitive latency or severe motion blur with frame-based sensors and the scarcity of electric energy during spaceflight missions all call for further investigation in this regard.

### A.2.2 Principles of event cameras

An excellent example on which to illustrate event camera principle is the Dynamic Vision Sensor (DVS) of Lichtsteiner et al. (2008). According to them, the event camera produces a pixel every time it registers a change in logarithm of intensity $I$ larger than a predefined threshold $\theta$

$$\Delta log(I) \geq \theta \tag{A.1}$$

After an event is produced the reference intensity is reset. Each pixel fires events completely independently and the result is an asynchronous stream. The event produced at a pixel $\hat{x}, \hat{y}$ is defined as follows with $t$ being a timestamp and $p$ the polarity (-1 or 1)

$$e = [\hat{x}, \hat{y}, t, p] \tag{A.2}$$

The Figure A.3 outlines functional similarities between the pixel circuitry of an event camera and biological retina. If the reader refers back to the Section A.1 they can see that the principle of operation of event camera and biological retina bears a strong similarity. This similarity can be to a point seen in the circuitry of individual pixels which contain functional blocks similar to photoreceptor, bipolar and ganglion cells. As described by Lichtsteiner et al. (2008) each pixel has a logarithmic photoreceptor circuit measuring light intensity. This photoreceptor is connected to a differentiating circuit resembling in function – as Posch et al. (2014) points out – the bipolar cell which activates if the logarithmic increment or decrement in intensity reaches certain threshold. According to Lichtsteiner et al. (2008) outputs from the differentiating circuit are then passed to the so-called Address-Event interface which interprets them as events and passes them to other devices using Address-Event Representation (AER) in a manner very similar to ganglion cells according to Posch et al. (2014). Lichtsteiner et al. (2008) describes AER as a communication protocol which allows to easily connect different neuromorphic devices by multiplexing spikes on one side of the bus and demultiplexing it on the other.

When characterizing or modelling an event camera the so-called uniformity of response is an important figure of merit. According to Lichtsteiner et al. (2008) it can be expressed as a noise when the camera is observing a fixed pattern. Numerically it can be characterized using the so-called threshold mismatch $\sigma_\theta$ which represents the standard deviation of contrast threshold $\theta$.

## A.3 Review of currently available event cameras

The first practical sensor was the already mentioned DVS developed by Lichtsteiner et al. (2008), which can be seen in the Figure A.4. The DVS parameters can be seen in the Table A.1. Serrano-Gotarredona et al. (2013) build up on this work and developed the so-called sensitive Dynamic Vision Sensor with order of magnitude lower noise when observing a fixed pattern and lower latency.

A very interesting development is the Dynamic and Active Pixel Vision Sensor (DAVIS) by Brandli et al. (2014) which features both characteristics of event cameras and frame-based cameras. In DAVIS the trigger mechanism of asynchronous events and synchronous frames share a photodiode, but do not affect each other in function. As can be seen from the Section D.3.1 this is very beneficial when developing algorithms to process event data as it allows to characterize performance easily by use of the so-called photometric loss. Somewhat similar is the ATIS sensor of Posch et al. (2011) which also includes absolute intensity information.

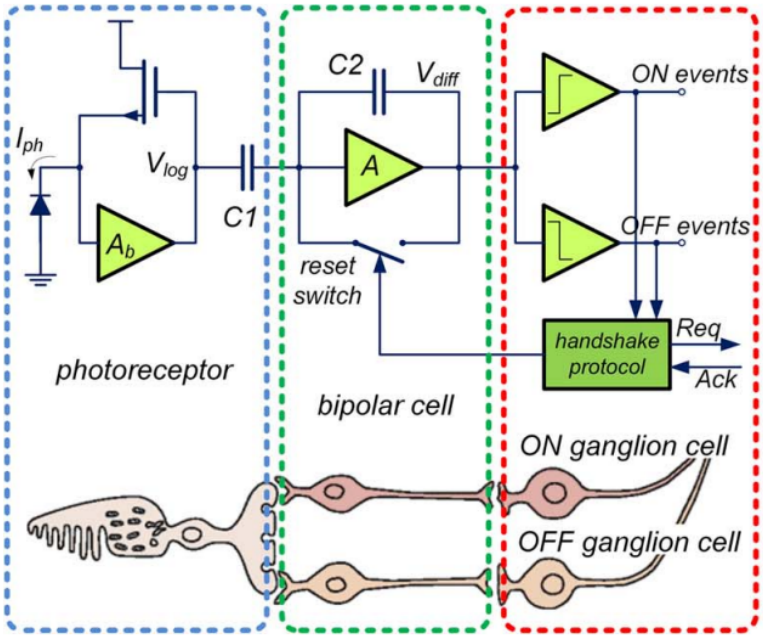



Figure A.3: Similarities between pixel circuitry of an event camera and a biological retina (image source: Posch et al. (2014)). here where taken

Table A.1: Parameters of DVS event camera by Lichtsteiner et al. (2008).

| **Characteristics** | |
|---|---|
| Array size | 128x128 |
| Power consumption | 24 mW @ 3.3V |
| Dynamic range | 120 dB |
| Latency | 15 $\mu s$ |
| Fixed pattern noise | 2.1% of contrast |

Figure A.4: The Dynamic Vision Sensor of Lichtsteiner et al. (2008) (image source: Hordijk et al. (2018))

Recently, event cameras were undergoing a strong increase in popularity and consequently also in a number of manufacturers who offer readily available solutions. Among the prominent ones is iniVation (who is currently responsible for the DVS and DAVIS sensors), Prophesee and Sony. As the amount of available sensors is now quite immense, it is considered unfruitful to go over all the options here.

## A.4 Strategies for generating event-based datasets

Obtaining event-based datasets may be a rather challenging task. Especially for a very specific purpose like investigating a planetary landing. Generally, however, two main options can be identified:

- Create a dataset using a physical event camera
- Convert a frame-based dataset to event-based one

Both of these options are discussed in the following subsections.

### A.4.1 Planetary landing event-based dataset by use of physical camera

While this may eventually change with an increased interest in the topic, there are currently no event-based data from real landings. However as elaborated in the Section 6.1 the camera will produce the same output if it is moving with a velocity $\mathbf{v}$ towards an object at distance $d$ and when the velocity is $\frac{\mathbf{v}}{k}$ and the distance $\frac{\mathbf{d}}{k}$ with $k$ being any positive real number. This allows to partially recreate the situation in the constrained conditions of a laboratory.

This was one of the approaches taken by McLeod et al. (2022). Both 2D and 3D printed surfaces were approached by an event camera mounted on a robotic arm with a depth sensor attached. A photo of the setup can be seen in Figure A.5.

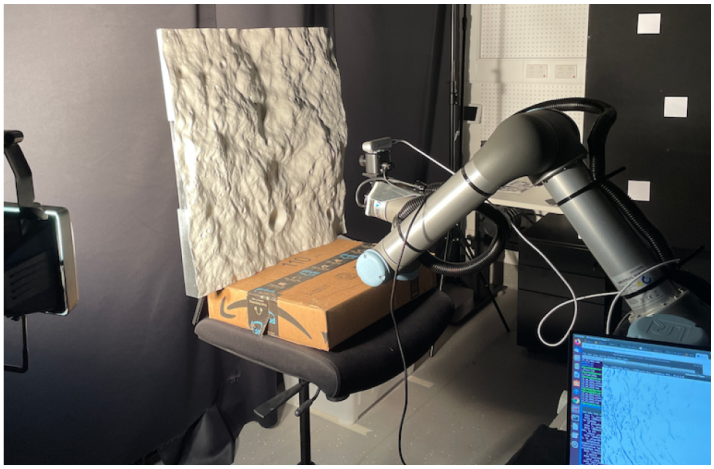


Figure A.5: Process of creation of event-based landing dataset by use of event camera mounted on a robotic arm and a 3D printed surface. Image taken from McLeod et al. (2022).

The approach of McLeod et al. (2022) has both advantages and disadvantages. The use of real event camera rules out one possible source of inaccuracies coming from poor simulation of the hardware. Obviously, effects of irradiation in space on the event camera are omitted. Roffe et al. (2021) conducted neutron irradiation experiments on event cameras. At radiation conditions similar to low Earth orbit a pendulum was first observed in a well-lit room with an event camera and then under the same conditions the camera behavior was observed with its lens covered. As such it is possible to measure the radiation noise and compare it to a signal. Signal-to-noise ratios of no less than 3.4 were achieved. As S. Zhang et al. (2020) explains lunar radiation environment is not much harsher than low Earth orbit. Event camera during landing is also likely to produce much stronger signal than when observing a pendulum as it will be moving fast above a rugged surface filling its full field of view. Based on these considerations it is deemed unlikely that radiation would have a significant influence on the event camera used during a landing maneuver. The much more significant disadvantage of this approach is its

incredibly high cost in time and resources.

It is noted that a similar approach was taken by Märtens et al. (2024) with Synthetic Lunar Terrain (SLT) dataset which presents horizontal trajectories over a terrain which is resemblant of the lunar south pole.

### A.4.2 Planetary landing event-based dataset by a conversion from a frame-based dataset

The option to convert a frame-based dataset seems to be much more popular. Unlike for event cameras there are actual and accessible frame-based camera data from lunar landings. Landing camera data from Chinese lunar landers Chang'e 4 and 5 have both been published and are available to researchers on the website of China's Lunar and Planetary Data Release System. Data from the navigation camera of the Japanese SLIM lander have been also publicly released.

The option to generate a synthetic dataset is also rather feasible. Planet and Asteroid Natural Scene Generation Utility (PANGU) by Martin et al. (2019) is capable of generating photorealistic images of planetary surfaces. PANGU seems to be a very popular option and is used for this purpose across many works (e.g. McLeod et al. (2022), Azzalini et al. (2023), Valette et al. (2010)). Another competing option is SurRender software (Brochard et al. (2018)) developed by Airbus with a very similar goal. It is difficult to compare PANGU and SurRender as relatively limited information is available to the author on SurRender. In the choice between the two, however, the license availability seems like the most important factor.

Azzalini et al. (2023) created a pipeline for generation of landing event-based datasets using PANGU and v2e event camera simulator elaborated in the next section.

## A.5 Event camera simulators

This section will discuss different methods of simulating pixel behavior when converting from frame-based to event-based dataset. Three main approaches to simulating the pixels were observed - deterministic models, stochastic models and learning-based methods. Firstly, behavior of a pixel will be introduced and then each of the options discussed.

### A.5.1 Pixel behavior

The simplest model of pixel behavior just defines ON and OFF thresholds $\theta_{ON}$ and $\theta_{OFF}$ and generates events according to the following rule with $L(\hat{x}, \hat{y}, t)$ being the log intensity

$$\begin{aligned} L(\hat{x}, \hat{y}, t_1) \geq \theta_{ON} \;&\rightarrow\; e(\hat{x}, \hat{y}, 1, t_1) \\ L(\hat{x}, \hat{y}, t_1) \leq \theta_{OFF} \;&\rightarrow\; e(\hat{x}, \hat{y}, -1, t_1) \end{aligned} \tag{A.3}$$

Such a model, however, completely ignores pixel non-idealities. For the purpose of describing real pixel behavior it is beneficial to divide the circuit into the parts shown in Figure A.6 as Joubert et al. (2021) suggest.

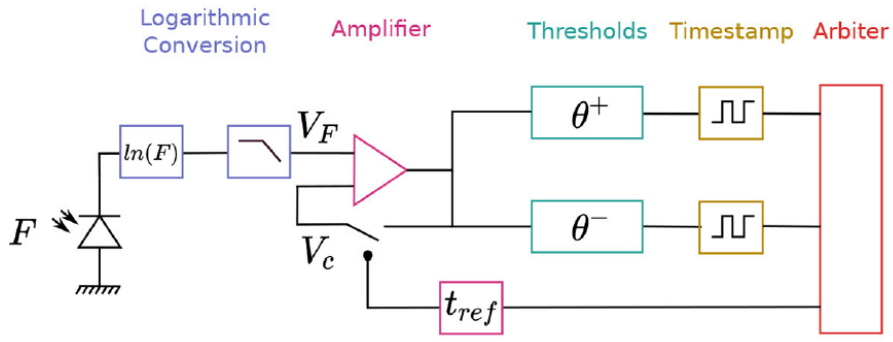



Figure A.6: Model of a pixel circuit for noise simulation (image source: Joubert et al. (2021), modified).

The quantal nature of the incoming photons is already the first source of non-ideality as it produces the so-called shot noise. Shot noise will manifest as random fluctuations in intensity of the incoming light. Hu et al. (2020) also make the observation that shot noise can have very significant effects on event cameras in low lighting conditions.

The photodiode in the photoreceptor part of the circuit produces current proportional to the light power. However as Hu et al. (2020) point out, even when light is absent the dark current still flows through the photodiode. According to Hu et al. (2020) this dark current will have rather small impact in good lighting conditions, it however becomes a significant factor under low illumination. Unusually high dark current can also be the cause of the so-called hot pixels. These are defective pixels which fire at an abnormal rate.

Lichtsteiner et al. (2008) also point out that the pixels have limited analog bandwidth and as such act as second-order low-pass filters with time constant indirectly proportional to the light level. Therefore this effect becomes again much more pronounced in low lighting conditions.

Further issues occur in the amplifier part of the pixel circuit. The amplifier transforms the photoreceptor output such that the current brightness value can be compared to the memorized brightness value at the thresholds. It suffers, however, from the so-called leakage current due to which the memorized brightness value decreases over time as explained in Hu et al. (2020).

The values of thresholds $\theta_{ON}$ and $\theta_{OFF}$ are actually not constant either due to the effect called transistor mismatch as stated in Lichtsteiner et al. (2008). Lastly, the real pixel has also a non-zero latency to fire an event.

### A.5.2 Deterministic simulators

Term deterministic simulator may be thought to be misleading, because some of the simulators described in this section indeed use randomness to generate the results. However compared to what we call the stochastic simulators they use a model with some random disturbances rather than directly sampling the event timestamps and polarities from a probability distribution.

Simulators of Kaiser et al. (2016) and Mueggler et al. (2017) use an ideal pixel and thus fall in this category. Somewhat more advanced approach is taken by ESIM (Rebecq et al. (2018)) and Vid2E (D. Gehrig et al. (2020)) as the mismatch of the threshold $\theta$ is simulated by assuming it to be normally distributed as $\theta \sim N(\mu(\theta), \sigma(\theta))$.

First serious attempt to model multiple pixel non-idealities is v2e (Hu et al. (2020)). Apart from the threshold mismatches other phenomena are included as shown in Figure A.7. Firstly the shot noise is modelled as a Poisson process. The v2e also introduces a model of dark current and simulates a second-order low-pass filter with a time constant being a function of light intensity. Such a filter however, as explained in Joubert et al. (2021), does not allow for analytical solutions to event timestamps and the current solving method makes the v2e output dependent on rendering rate. Leakage current is also modelled by decreasing the memorized brightness value in discrete time steps.

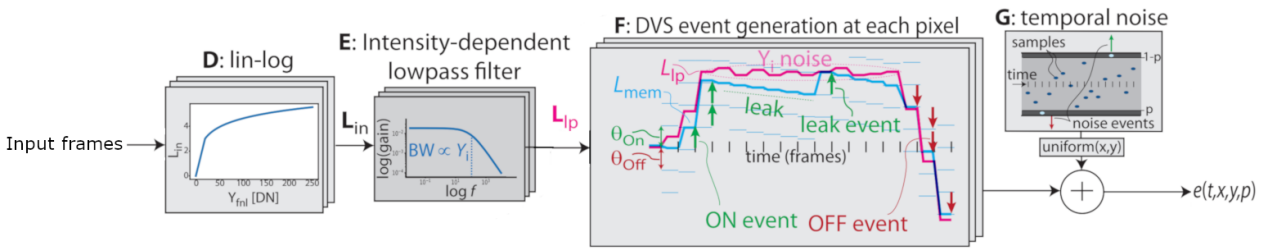



Figure A.7: Pixel model used by the v2e event camera simulator, first intensity frames are converted to log values (D) and passed through a second order low pass filter (E). These input values are fed into a discrete time model of the amplifier and thresholds (F). Lastly temporal noise (G) such as shot noise is mixed into the final output. (image source: Hu et al. (2020), modified).

IEBCS (Joubert et al. (2021)) is a simulator which tries to address certain v2e deficiencies especially the dependency between rendering rate and the resulting events. It approximates the second order low-pass filter

in v2e with a first order low-pass filter with latency. It shows that this approximation holds well for a typical event camera pixel and it allows to solve for the exact timestamps. Moreover it introduces a purely stochastic method of generating temporal noise events (such as due to shot noise or dark current) where each pixel simply draws noise event timestamps from a previously collected cumulative distribution. This cumulative distribution will indeed vary when an order of magnitude change in the light intensity on the pixel occurs. For that case IEBCS offers three cumulative noise distributions (for 0.1 lux, 161 lux and 3000 lux) and switches between them according to the light conditions on the pixel.

### A.5.3 Stochastic simulators

DVS Voltmeter (Lin et al. (2022)) is a simulator which tries to reduce the event simulation to a purely stochastic process. Between each two interpolated frames two parameters $\mu$ and $\sigma$ are calculated for the pixel based on the average brightness $\bar{L}$, brightness change $\Delta L$ and the time step $\Delta t$ as follows with coefficients $k_1$ to $k_6$

$$\begin{aligned} \mu &= \frac{k_1}{\bar{L} + k_2} \frac{\Delta L}{\Delta t} + k_4 + k_5 \bar{L} \\ \sigma &= \frac{k_3}{\bar{L} + k_2} \sqrt{\bar{L}} + k_6 \end{aligned} \tag{A.4}$$

The coefficients $k_1$ to $k_6$ need to be empirically fitted to a camera and rough representation of the condition. Authors of Lin et al. (2022) provide a recommended calibration method for DAVIS cameras (which provide both events and intensity frames) using regression which is however rather laborious and requires some manual tuning.

With the parameters $\mu$ and $\sigma$ known, the probability of next event on a pixel being an ON event is expressed as a function of the respective parameters and the ON and OFF event thresholds $\theta_{ON}$ and $\theta_{OFF}$. The length of time interval between two consecutive events on the pixel $\tau$ (which can be used to fully determine the timestamp) is then drawn from an inverse Gaussian distribution

$$\begin{aligned} \tau_{ON} &\sim \mathrm{IG}\left( -\frac{\theta_{ON}}{\mu}, \frac{\theta_{ON}^2}{\sigma^2} \right) \\ \tau_{OFF} &\sim \mathrm{IG}\left( \frac{\theta_{OFF}}{\mu}, \frac{\theta_{OFF}^2}{\sigma^2} \right) \end{aligned} \tag{A.5}$$

### A.5.4 Learning-based simulators

Another approach to the pixel simulation is purely data driven. There are two available models, EventGAN (Zhu et al. (2019)) and V2CE (Z. Zhang et al. (2023)).

EventGAN uses a Generative Adversarial Network framework to generate events from a pair of input images. An important drawback of EventGAN is that it does not attempt to generate accurate timestamps of the events. Instead of that, it partitions the spatio-temporal domain into voxels and generates only the events-per-voxel counts. This has its advantages as it results in substantially simpler loss functions when training. It is, however, impossible to use EventGAN with any processing algorithm which is sensitive to the exact timing of events.

V2CE uses a U-Net-like convolutional neural network (please see Section D.3 for further details on the concept) to simulate the occurrence of events with accurate timestamps.

V2CE first predicts the occurrence of events in spatio-temporal voxels and then refines the timestamps using a stochastic timestamp prediction model. The neural network used by the V2CE can indeed take as an input multiple frames. As such it can be trained to become motion-aware and predict accurately events even in cases of highly non-linear motion.

# Appendix B

# Trajectories for landing on an airless body

Airless bodies such as moons, comets or asteroids pose a unique challenge for designers of lander vehicles. As there is no medium which would help to slow the vehicle down, propulsion system must achieve this velocity change alone. It is therefore very important what strategy is chosen for the control of the landing such that minimal amount of fuel can be carried. Optimal control theory allows to find such control laws and the corresponding trajectories. In this chapter, the general aspects of lunar landing will be first introduced in Section B.1. Then trajectory optimization will be discussed in Section B.2. Lastly, Section B.3 will introduce modern methods which can be utilized during such undertakings.

## B.1 Lunar landing

This section will briefly discuss the main considerations connected to soft lunar landing. Firstly the lander vehicles will be briefly discussed followed by a general discussion on the landing trajectory phases and strategies.

### B.1.1 Lunar lander vehicles

The author was capable of finding mentions of 56 soft landing attempts of which 22 were successful (as a side-note it is quite astounding that the number of successful landings raised from 20 to 22 only during the period of this research). All the vehicles share certain similarities.

Firstly, the author was unable to find a vehicle which would not consist of a payload section coupled with a retrorocket and an attitude control system. As an example a simple diagram of the Surveyor lander is presented in Figure B.1. Even the earliest attempts, like for example the landers of the unsuccessful Ranger program, complied to this structure as shown by Ball et al. (2007). Not all the retrorockets were however possible to throttle depending on the circumstances, for example the successful Surveyor missions used also solid retro-motors as stated by Thurman (2004).

Majority of the landers also try to land at very low velocities, Ball et al. (2007) lists 0.9 m/s for the Apollo missions and well below 10 m/s for other legged landers. However, as noted in Ball et al. (2007), landers equipped with energy absorption mechanism are an exception from this rule. Ranger or Luna 9, which may serve as examples of such landers, used landing speeds that were order of magnitude higher. If a legged lander is considered though, it can be essentially seen as coming to full standstill during the landing from a trajectory optimization viewpoint.

### B.1.2 Landing trajectories

As pointed out by Ball et al. (2007), there are generally two strategies which can be used for lunar landing. The vehicle can either enter the landing from a closed orbit around the Moon or from a hyperbolic orbit which passes close to the surface. However the general principle remains the same. The vehicle must first get rid of enough velocity to get on a trajectory which passes through lunar surface and then slow down enough to perform a soft landing.

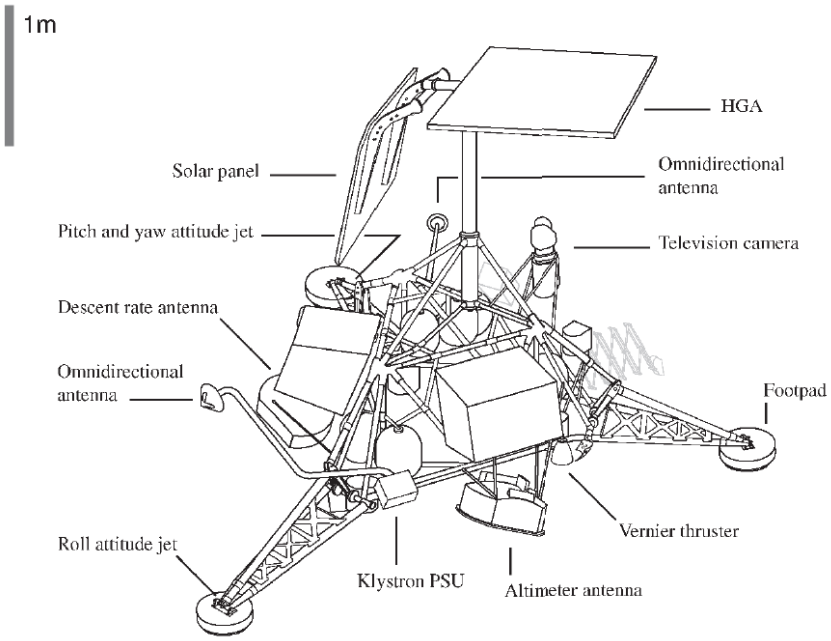



Figure B.1: Simple diagram of the Surveyor lander (image source: Ball et al. (2007)).

Such landing usually consists of multiple phases. The most common division includes braking, pitch up, approach and final or vertical descent used e.g. by A. Lee et al. (2010). A typical landing trajectory split up into these phases can be seen in Figure B.2. According to A. Lee et al. (2010), the orbital velocity is efficiently reduced during the braking phase. A pitch-up maneuver is done to allow good visibility of the landing site. In the approach phase the rest of the orbital velocity is nulled and final decisions are made. Lastly in the final descent phase the vehicle acts against the gravity to softly touch the surface.

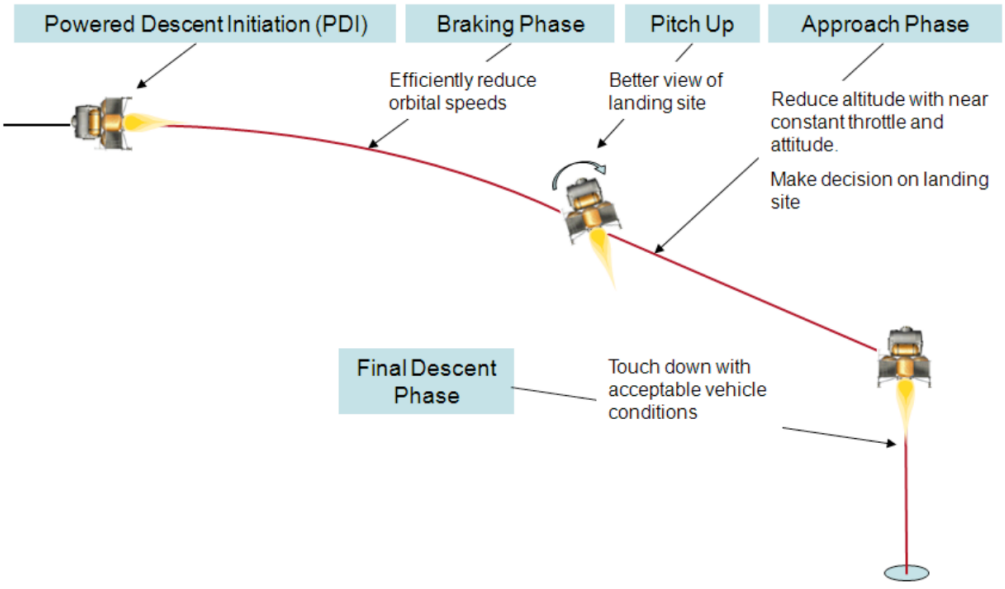



Figure B.2: Proposed phases of a lunar landing for the cancelled Altair vehicle (image source: A. Lee et al. (2010)).

As an example it is stated by Chu et al. (2017) that the Apollo braking phase began at circa 15 kilometers above the surface of the Moon and continued to altitude above 2 kilometers. There the pitch-up happened and approach phase continued until reaching roughly 150 meters above the surface. From there the vehicle proceeded with a final descent. The point of initiation of the approach phase is often called as high gate and the point where it transitions into final descent as low gate.

Lunar landing trajectories have evolved over time. Ball et al. (2007) notes that the early landers such as Surveyor had very simple guidance algorithms which essentially amounted to what is called the gravity turn as the vehicle simply achieves soft landing by always applying the thrust in the direction exactly opposite to the spacecraft trajectory. Such solution is however extremely inefficient. Therefore since the Apollo program near-optimal or optimal guidance laws have been studied, proposed and used such as the one used by Apollo described in Klumpp (1974).

## B.2 Trajectory optimization

Trajectory optimization problems seek to minimize or maximize a certain objective over a trajectory which meets a set of constraints.

The field of optimal control on spacecraft is very broad. This is because most of the methods consist of sub-methods which can be often rather freely matched together resulting in a great variety of combinations. Based on Betts (2010), Conway (2010), Leonard and Van Long (1992) and Klumpp (1974) distinction is now made between the main philosophies. The methods can be roughly divided into:

- Optimal and near-optimal
- Closed-loop and open-loop
- Direct and indirect

Optimal control problems are generally fairly computationally expensive. Therefore it was not uncommon in real-life applications to use some heuristics or perhaps a smart modification of the problem to reach a special case which is not actually optimal, but usually comes very close to optimality. Such near-optimal methods are nowadays mostly considered heritage as they were rather popular with low-performance old computers though.

A concise explanation of difference between open-loop and closed-loop methods is given by Leonard and Van Long (1992). Open-loop methods relate the controls to time. On the contrary closed-loop methods relate the controls to the state variables. For the purpose of this study open-loop methods are considered only as the interest lies in the trajectory itself rather than the control problem.

As Betts (2010) states optimal control problem solving methods can be divided into indirect ones (or optimize-then-discretize methods) and direct ones (also known as discretize-then-optimize methods). The indirect methods use calculus of variations to construct necessary and sufficient optimality conditions and then find a solution which satisfies them by numerical or in extremely simple cases analytical means. On the other hand direct methods simply try to numerically minimize the objective function. According to Betts (2010) the main practical difference is better accuracy of indirect methods countered by their larger sensitivity to initial conditions and more mathematical demands on the user. While this is not a rule, mostly the heritage methods rely on indirect approach while more modern solutions use the direct approach.

## B.3 Modern methods for fuel-optimal landing

The trajectory optimization problems in the modern times are typically solved numerically using the so-called non-linear programming. If e.g. rotational dynamics of the vehicle are to be considered this becomes almost a necessity. This section will shortly discuss the relevant methods.

### B.3.1 Non-linear programming

As Luenberger (2003) states, non-linear programming in general concerns solving problems of the following type with m inequality constraints and p equality constraints

$$\begin{aligned} &\text{minimize} && J(t) \\ &\text{subject to} && f_i(t) \leq 0, \quad i = 1, \ldots, m \\ &&& h_i(t) = 0, \quad i = 1, \ldots, p \end{aligned} \tag{B.1}$$

As Malyuta, Yu, et al. (2021) explains, direct powered landing trajectory optimization problem falls within this category. However the exact formulation of the optimal landing problem has huge implications on how this problem should be solved. Some simple formulations will involve minimizing convex functions over convex regions and thus can be solved using convex optimization. This is of great benefit because as Malyuta, Reynolds,

et al. (2021) explain, such algorithms are bound to find global optimum in polynomial time. On the other hand formulations involving rotational dynamics or other more advanced cases will typically require use of general non-linear program solvers.

### B.3.2 Convex optimization

As Malyuta, Reynolds, et al. (2021) note, if a lander is considered to be a lumped mass and all the state and control constraints are one-sided inequalities, the resultant problem can be solved by convex optimization. While very simplistic, this can be a good representation of a lander behavior. As Açıkmeşe et al. (2012) note, for most vehicles the attitude and translational dynamics behave as essentially decoupled and inclusion of attitude dynamics typically has a limited effect on the result of the optimal control problem. Moreover, if the thrust vector is considered to be fixed with regards to the vehicle, it is still possible to recover the optimal attitude information.

Açıkmeşe and Polen (2007) note that this algorithm has one disadvantage, namely the impossibility to incorporate a lower bound for the engine thrust. Many liquid engines cannot be throttled down completely when started due to combustion instabilities, as Malyuta, Reynolds, et al. (2021) explain. Therefore Açıkmeşe and Polen (2007) convexify the thrust constraint by utilizing the so-called slack variable $\Gamma(t)$. As Malyuta, Reynolds, et al. (2021) explain, the double-sided thrust constraint which forms a non-convex annulus is rewritten into two constraints which form a convex volume in the following manner

$$\begin{aligned} &0 \leq \rho_1 \leq u_t(t) \leq \rho_2 \\ &0 \leq \rho_1 \leq \Gamma(t) \leq \rho_2, \mid u_t(t) \mid \leq \Gamma(t) \end{aligned} \tag{B.2}$$

Such reformulated problem is shown by Açıkmeşe and Polen (2007) to have the same global optimum as the original problem. The smart use of lossless convexification can be beneficial in other cases as well and Açıkmeşe and Polen (2007) and Açıkmeşe et al. (2012) also add possibility to use glideslope and pointing constraints which would be otherwise non-convex. As Blackmore (2016) and Açıkmeşe et al. (2012) reveal, this method is popular for in-flight use and e.g. the SpaceX rockets or the Masten Xombie powered landing prototype use the real-time version of the algorithm.

### B.3.3 General non-linear programming

As stated previously, more general problems which e.g. involve attitude dynamics are not necessarily convex and typically cannot be convexified. They can still be often solved by numerical methods. There are however no guarantees of a global optimum reached in polynomial time which are inherent to the convex optimization. It is considered redundant to discuss each of these methods in great detail as there is a very large number of them, often with similar properties. This subsection will therefore include only a short discussion of perhaps the most popular option in context of landing optimization which is the sequential convex programming, a method which now also enjoys frequent practical use in real-time optimal control applications (it is e.g. used during the SpaceX Starship rocket "belly flop").

As Duchi (2018) explains, sequential convex programming is an optimization method built on convex optimization. It essentially divides the problem into convex and non-convex parts, replaces the non-convex parts with their locally convex models and then treats the problem in manners standard for convex optimization. It is a well-described method in context of powered landing with prominent examples being Mao et al. (2016), Bonalli et al. (2019), Reynolds and Mesbahi (2020) and Wang and Grant (2016). It is considered beyond the scope of the literature study to offer a deep dive into this topic as a comprehensive review would be more than a work on its own. For a mere practitioner there exists an excellent Sequential Convex Programming toolbox (Malyuta et al. (2022)) which implements the most important algorithms and offers easy to follow tutorials on optimizing rocket trajectories using the respective method family.

# Appendix C

# Artificial neural networks in the context of event-based optical flow

This chapter will discuss the principles and structure of neural networks as far as they are relevant to the subject matter of the literature study. First, Section C.1 will introduce artificial neural networks in general. Then, Section C.2 will present the spiking neural network paradigm.

## C.1 Artificial neural networks

Artificial neural networks are a novel computing paradigm which has lately revolutionized multiple areas of research and industries. As artificial neural networks are an extremely broad topic, this section will only focus on what is very relevant to event-based optical flow. In case of further interest, reader is encouraged to study either Aggarwal (2018) or Nielsen (2018) (based on which this section is formed).

### C.1.1 Basic principles of artificial neural networks

As Aggarwal (2018) states a neural network is a directed acyclic graph. The edges of the graph are assigned weights and the computations in each node of the graph are dependent both on the incoming edge weights and the values of nodes connected to it by these respective edges. Each node is called a neuron and has its own activation function which takes the edge weights and input values as arguments to produce a single output. Importantly, both the edge weights and activation functions are at least partially learnable from training data. This is to a point inspired by the function of a biological brain which is touched upon in the Subsection C.2.1. An illustration of a simple network structure is given in the Figure C.1.

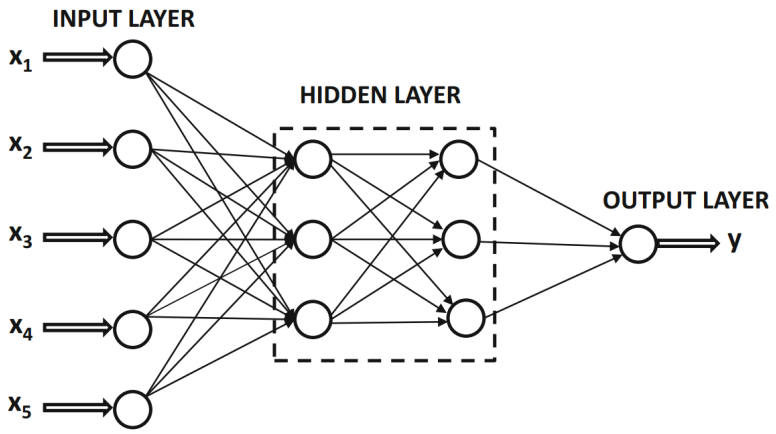



Figure C.1: Illustration of a simple neural networks structure. There is an input layer which serves as an input interface. The layers which are nor input, nor output are referred to as hidden. They propagate the information to an output layer. Image taken and modified from Aggarwal (2018)

As Nielsen (2018) states such an arrangement allows the networks to compute any continuous function if appropriate architecture, weights and activation functions are used. This makes them excellent candidates to capture

highly non-linear relationships.

As put by Aggarwal (2018) the layers can be either fully connected (each neuron in layer A is connected to every neuron in layer B), sparsely connected (not every neuron in layer A is connected to each neuron in layer B) or recurrent connections can appear (which allow previous outputs from neurons in layer A to be connected to layer A).

The choice of activation function has a major influence on the function of the neural network. According to Nielsen (2018) these are typically differentiable functions in form of $a(\mathbf{x}_{input}, \mathbf{w}, b)$ with $\mathbf{x}_{input}$ being the input vector, $\mathbf{w}$ being the vector of input weights and $b$ being the learnable bias parameter. Output of such function in computer applications is typically a floating point number. The existence of spiking neural networks described in the Section C.2 – which have digital activation functions – leads to a distinction in which the traditional artificial neural networks are also called analog neural networks.

### C.1.2 Backpropagation

As Nielsen (2018) states, the full power of artificial neural networks can only be utilized if they are sufficiently large and contain a significant number of layers. To train such a network would be very challenging were it not for a so-called backpropagation algorithm which allows to learn the weights and other terms efficiently. The backpropagation algorithm itself is vastly too complicated to be elaborated here and the reader, if interested, is recommended to see Nielsen (2018) but the basic principle will be shown. A cost function $C(o_a, o_d)$ where $o_a$ is the actual output of the neural network and $o_d$ is the desired output is defined. As Nielsen (2018) explains backpropagation calculates the partial derivatives $\frac{\delta w}{\delta C}$ for all the parameters $w$ in the network efficiently using chain rule. This allows to calculate the gradient $\nabla C$ with low computational cost. If the gradient is known then the parameters can be optimized using methods such as gradient descent.

## C.2 Spiking neural networks

Spiking neural networks handle incoming data differently than traditional artificial neural networks. They try to more closely mimic the biological processes in order to increase efficiency. This section will describe the biological background of such networks, their basic principles, models of neurons used in such networks, learning processes in this context and it will lastly shortly discuss what this novel computing paradigm means for hardware.

### C.2.1 Biological background

The inspiration for spiking neural networks comes from the function of a biological brain. While it would be unnecessary to go into too much detail, it is considered a good idea to review the main principles which are relevant to the spiking neural networks. The following short description of neural dynamics originates from Gerstner and Kistler (2002).

The basic functional unit of the brain is the neuron which can be divided into three parts by function – dendrites, soma and synapses. The dendrites serve to receive the input signals, soma does the processing of the signals and the synapses handle the output signals. A neuron which is sending the signal is known as pre-synaptic neuron and by the same logic the receiving neuron becomes the post-synaptic neuron. A simple diagram showing these components can be found in the Figure C.2.

The aforementioned neuronal signals are very short electrical pulses often called spikes (hence the name of spiking neural networks). The arriving spikes affect the membrane potential of the soma. The membrane potential is simply the voltage difference between the inside of the cell and the surrounding fluid. An arriving spike can either increase or decrease the membrane potential and according to the nature of the change it is either called an excitatory or an inhibitory spike.

If the membrane potential reaches a certain threshold a neuron will fire a spike. After firing, the membrane potential first strongly decreases while it undergoes hyperpolarization to progressively reach the rest potential later. This phenomenon ensures that even if strongly excited, the neuron cannot immediately fire again as it is in a state of refractory period. As a result, sufficient temporal distance between individual spikes is always kept and spikes coming out of the neuron are never ambiguous.

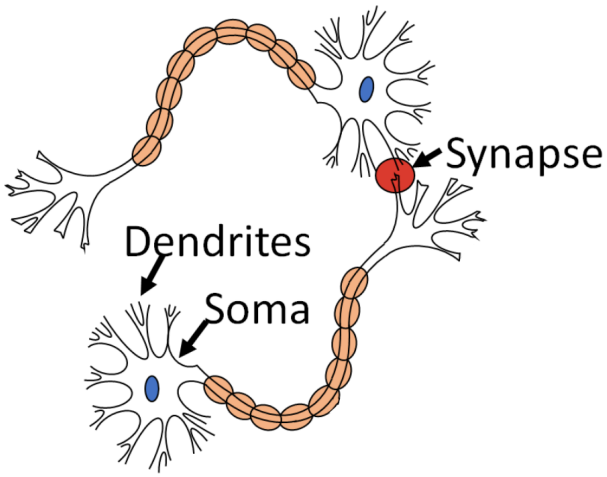



Figure C.2: Simple schema of a pre-synaptic and a post-synaptic biological neuron showing the main functional components. Image taken and modified from Maheshwari et al. (2022)

### C.2.2 Spiking neural network principles

Following these biological principles led to a novel type of neural networks which more closely mimic the function of biological neurons. Instead of using floating point values the neurons only communicate by sending spikes at a certain point in time. The following discussions will touch encoding, neuron models and learning in such networks.

There is currently a debate on how the input information to a spiking neural network should be encoded. According to W. Guo et al. (2021), there are four larger groups of encoding strategies. They are known as rate coding, time-to-first-spike coding, phase coding and burst coding. As M.-H. Guo et al. (2022) state, the rate coding is the simplest one. If the reader imagines an input to a specific neuron as a value, rate coding will create a spike train with a firing rate corresponding to the very value of the input. In case of time-to-first-spike, the exact timing of the first spike will be indirectly proportional to the value. Phase coding converts the input into a binary representation and then fires the spikes according to the sequence of ones and zeros. Lastly, burst coding sends bursts of spikes and the number of spikes contains the necessary information. These coding schemes are illustrated in the Figure C.3.

### C.2.3 Neuron models for spiking neural networks

There exists a wide variety of ways in which neurons can be modelled in spiking neural networks. They bring a range of computation costs, network performance and biological plausibility.

A very biologically plausible neuron model was created by Hodgkin and Huxley (1952) who studied squids to come up with a representation of the neuron as an electrical circuit with variable conductances. The characteristics of such a cell are fully described by four ordinary differential equations (therefore it is often called as four-dimensional). This leads to very significant computational costs.

The high computational complexity led to the development of neuron models with higher levels of abstraction which try to capture the overall behavior of neurons rather than the exact mechanics as explained in Maass and Bishop (1999). A very important part in this area is played by the family of integrate-and-fire neuron models. Such models are much older than the biologically plausible ones as they were first proposed more than hundred years ago according to Brunel and van Rossum (2008). The key element of all the integrate-and-fire models is the assumption that the neuron behaves like a capacitor which fires a spike when a certain membrane potential threshold is reached. In its simplest form, such a model takes the form as shown in the Equation C.1 with $V(t)$ being the membrane potential, $C_m$ being the membrane capacity and $I(t)$ being the input current. Note that now the characteristics of the cell are described by only one differential equation leading to superior computational costs (therefore it is often called as one-dimensional).

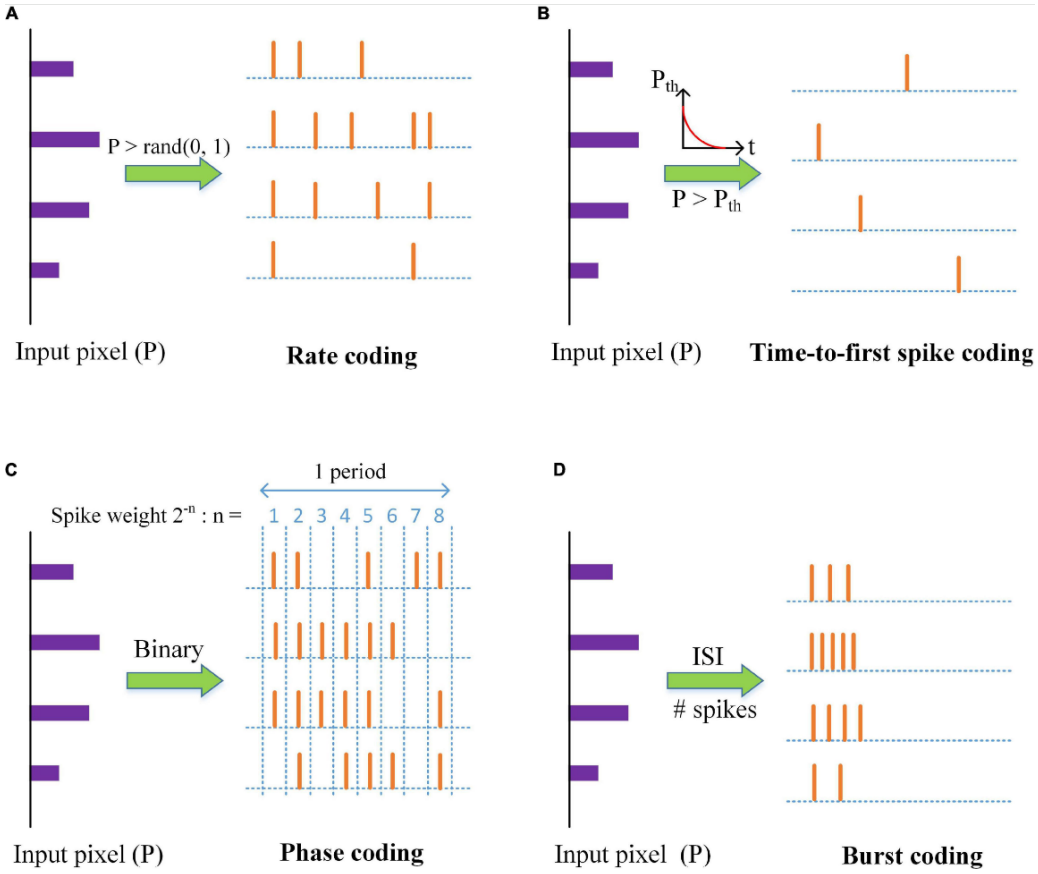



Figure C.3: An illustration of coding schemes for input to spiking neural networks. An example is given for four different input pixels, each with a float value $P$. In rate coding the spike firing rate is proportional to value of $P$. For time-to-first-spike coding the time of first spike will be indirectly proportional to P. Phase coding transfers the value of $P$ to a binary representation and then transmits the spikes as ones and non-spikes as zeros. Lastly burst coding fires bursts of size proportional to $P$. (Image source: W. Guo et al. (2021))

$$C_m \frac{dV(t)}{dt} = I(t) \tag{C.1}$$

Very often, the integrate-and-fire model is extended into a leaky integrate-and-fire model as described in Gerstner and Kistler (2002). In such a case, an additional resistor with resistance $R_m$ is added in parallel to the capacitor as shown in the Figure C.4, and the dynamics follow the Equation C.2. The advantage of such a neuron compared to the simple integrate and fire model is that if no spikes are arriving the voltage eventually drops as the current is "leaking" through the resistor. This allows to better utilize the temporal characteristics of the incoming spikes.

$$C_m \frac{dV_m(t)}{dt} = I(t) - \frac{V_m(t)}{R_m} \tag{C.2}$$

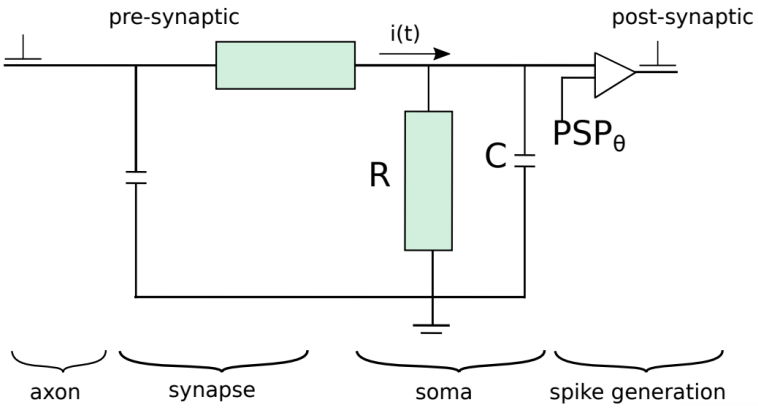



Figure C.4: An illustration of a circuit of a leaky integrate-and-fire neuron (image source: Lobo et al. (2019)).

The integrate-and-fire family models exist in a large variety. One example are adaptive integrate-and-fire neurons which are elaborated in Sung and Kim (2020). The key property of the adaptive neurons is their ability to adapt to specific scenarios over several spikes as they do not lose all the memory when they fire.

The model presented by Izhikevich (2004) can be seen as an intermediate step between the one-dimensional integrate-and-fire models and the four-dimensional model of Hodgkin and Huxley (1952). It is a two-dimensional model that was derived from Hodgkin and Huxley (1952) using bifurcation methodologies in order to provide very high level of biological plausibility while keeping reasonable computation costs.

### C.2.4 Learning in spiking neural networks

Spiking neural networks face several unique constraints during learning. As the activation functions are not differentiable, it is impossible to make use of the usual combination of backpropagation with stochastic gradient descent. This subsection will discuss some prominent alternatives.

A fairly straightforward option is what Eshraghian et al. (2021) call shadow learning. An analog neural network with the exactly same structure is trained and converted to a spiking neural network. Out of the multiple options a conversion of analog activation functions into spike rates as in Diehl et al. (2016) can be pinpointed. According to Eshraghian et al. (2021), however, shadow training methods suffer from loss of temporal dynamics as the analog neural networks lack the means to handle such inputs.

Possibly much more attractive option are modifications of the backpropagation algorithm for use with spiking neural networks. One option is the so-called SpikeProp introduced by Bohté et al. (2000). SpikeProp calculates the error with respect to the differentiable spike times instead of the non-differentiable activation functions. This allows to train the network with a fairly common backpropagation algorithm. However, as Eshraghian et al. (2021) points out, the algorithm is prone to let neurons "freeze" as if they never spike, they can never learn. Thus, according to Eshraghian et al. (2021), the algorithm is very sensitive to weight initialization and the required initial weights may lead to suboptimal results.

The most common way at the moment is the use of surrogate gradients. As described by Neftci et al. (2019), in such method the activation function differs during the forward and backward pass. In the forward pass, it is the Heaviside function commonly used in spiking neural networks, but during the backward pass it is approximated by a differentiable function such as the sigmoid. According to Eshraghian et al. (2021), with such a modification, training can be done by using methods like backpropagation through time which are widely known from recurrent analog neural networks. The backpropagation through time essentially unfolds the network into the past, and do not calculate the gradient also with respect to a certain amount of previous timesteps which incorporate the influence of prior events as shown in Werbos (1990).

All the above mentioned methods are primarily popular with supervised learning. There are more exotic ways of training a spiking neural network which are largely but not exclusively connected to unsupervised learning. Spike-timing-dependent-plasticity is a famous example. This method comes from characterization of a biological phenomenon by Hebb (1949) which has been later paraphrased as "neurons which wire together, fire together" by Shatz (1992). As explained in Roy et al. (2019) the strength of connection between a pre-synaptic and a post-synaptic neuron is strengthened if the post-synaptic neuron fires after the pre-synaptic one. On the other hand it is weakened if the situation happens the other way around.

### C.2.5 Neuromorphic hardware

Spiking neural networks have a huge advantage in possible compatibility with the so-called neuromorphic hardware which completely differs from the common von Neumann architecture and ensures very low latency and power consumption. As discussed in Schuman et al. (2022), the biggest differences are massively parallel processing used in the neuromorphic hardware and communication with asynchronous spikes instead of blocks of binary data processed in a synchronous manner. These differences are summarized in the Figure C.5.

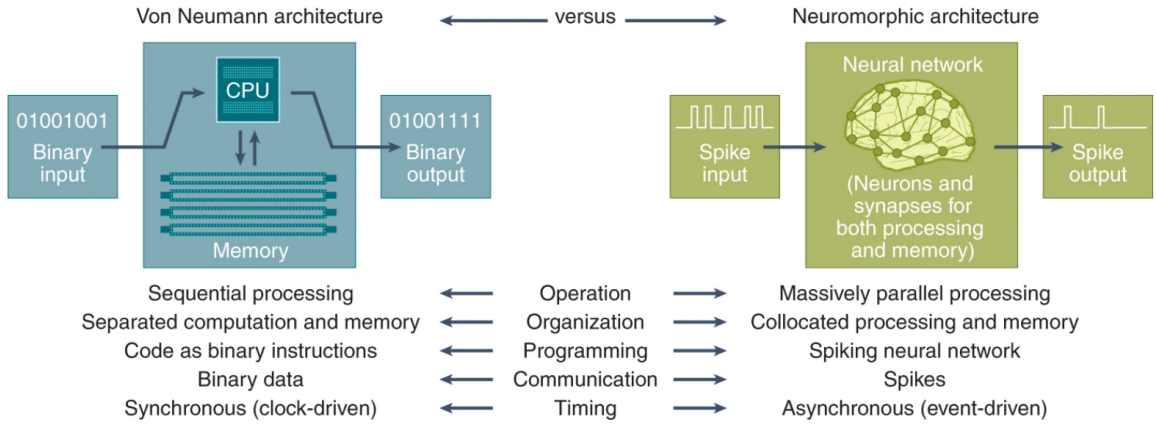



Figure C.5: Qualitative comparison of von Neumann architectures and neuromorphic architectures (image source: Schuman et al. (2022)).

It is not considered beneficial for this work to dive deep into this topic, so the overview is kept very brief. Some famous examples of a neuromorphic chip are Loihi by Davies et al. (2018) (well used in guidance, navigation and control applications for drones e.g. Paredes-Vallés et al. (2023), Valette et al. (2010)) or TrueNorth by Akopyan et al. (2015). It is very difficult to say that some operations or calculations are downright impossible to do on these devices, however there are some general preferences.

Cuadrado et al. (2023) point out that a good strategy to avoid issues during implementation is to adhere to the binary, spiking nature of the spiking neural networks as much as possible and preferably avoid e.g. use of floating point values where possible. In the specific case of Cuadrado et al. (2023) this for example leads to a replacement of bi-linear upsampling, which involves averaging and thus floating operations, with nearest-neighbor upsampling, which can be done in purely binary fashion.

For the case of neuromorphic computing in space it is very interesting to compare algorithms against each other in terms of computational cost. The varied nature of neuromorphic hardware sadly makes this very difficult. Bains (2021) published an opinion piece underpinning how easy it is to accidentally start comparing "apples with oranges" due to the wide variety of neuromorphic hardware and proposed that comparison schemes often work poorly on technologies with such a low level of maturity. The author could indeed not find any largely accepted measure for computational cost which would mirror e.g. the floating point operations (FLOPS) for

von Neumann architectures. This suggests that the work should rather focus on other performance measures than computational cost. However it may be interesting to see if some of schemes like the one proposed by Date et al. (2021) can be applied.

### C.2.6 Simulation of spiking neural networks

There is a number of ways in which spiking neural networks can be simulated on commonly used computers. A currently very popular language for prototyping and testing such networks is Python, because of its relative simplicity, familiarity in the engineering community and the vast amount of libraries which can be used. A large overview of such libraries is presented in Yamazaki et al. (2022). Probably the most interesting one for the purpose of this work is Norse (Pehle and Pedersen (2021)) and snnTorch (Eshraghian et al. (2021)) due to their support for variety of backpropagation and backpropagation-through-time algorithms and wide documentation.

Hagenaars and Paredes-Vallés (2023) is a repository which provides tools and guidance for easy simulation of spiking mechanics using classical machine learning libraries. The authors of Hagenaars and Paredes-Vallés (2023) make a point that this form of self-implementation may be a good option as it is often better to implement only the necessary features to remove additional complexity.

## C.3 Convolutional neural networks

Convolutional neural networks have become a prominent solution for a lot of computer vision problems. The situation is no different in the field of processing event camera outputs. This section will present a short overview of their principles and special considerations connected to spiking implementations.

### C.3.1 Convolutional neural network principles

Most neural networks are not particularly well-suited to handle very large volumes of grid-like data. Very soon such networks become almost untrainable due to the sheer amount of parameters which arise during such trainings. Therefore convolutional neural networks were developed to tackle these issues.

The main idea of convolutional neural networks is the use of a kernel (also known as a filter) to extract features spanning over multiple pixels from the data using convolution as explained in Venkatesan and Li (2018). This results in fewer neurons and connections leading to more manageable networks. Kernels are learnable matrices of values which are passed over the grid data. The result of convolution is simply a sum of products between all the numbers in the kernel and the convolution input they overlay. This is illustrated by the Figure C.6. Kernels can then effectively reduce dimensionality of the input and extract characteristic features.

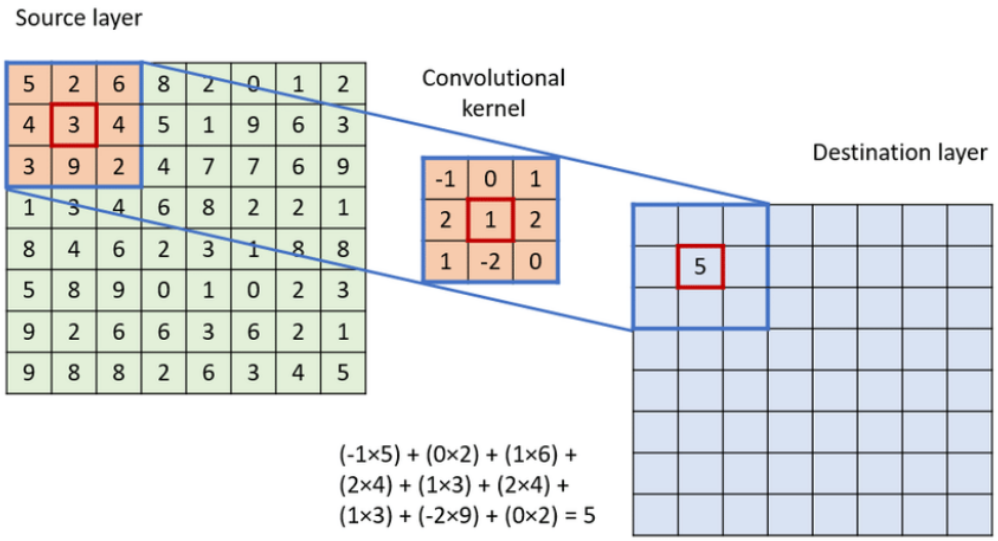



Figure C.6: An illustration of a two-dimensional convolution kernel. In this case a kernel is placed on top of the number three in the input layer. Summing all the products of kernel values with the underlaying values in the input layer leads to the final result of 5 which is written into the destination layer (often called feature map in context of convolutional neural networks). (Image source: Podareanu et al. (2019))

As Venkatesan and Li (2018) mention, convolution can have some special properties. For example it can be strided, which means that the convolution kernel skips over some of the numbers in the input layer with a regular step size. Another option is padded convolution, where it is made possible to pass the kernel over values on the sides and corners by feeding artificial values to the parts of the kernel which would be otherwise outside of the input.

Apart from convolution layers, convolutional neural networks typically feature the so-called pooling layers. As Ranjan (2020) points out, pooling is an operation which is supposed to reduce the size of the representation of the features while preserving the important information. The general idea of pooling is to take certain values in the input and represent them as a single value. The Figure C.7 represents two of the most common pooling methods, the maximum and average pooling.

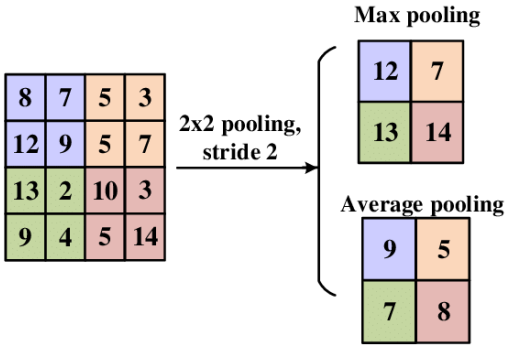



Figure C.7: Illustration of the pooling mechanism. Pooling with stride two is done on the input which means that two values are always skipped when moving the 2x2 pooling operator in each direction. Average pooling results in an average over the 2x2 window while maximum pooling simply takes the largest element. (Image source: Yingge et al. (2020))

Deep convolutional architectures typically consist of multiple convolutional and pooling layers. A typical example can be seen in the Figure C.8. In a characteristic setting, features are extracted from the input and then processed in fully connected layers. Note that some convolutional neural networks used e.g. for segmenting or optical flow estimation require a high resolution output. In such cases, a variety of upsampling methods can be used such as nearest neighbor interpolation.

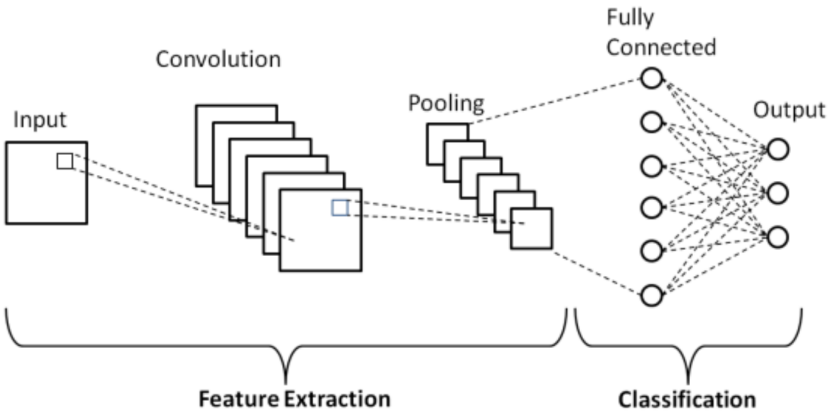



Figure C.8: Schematic diagram of a simple convolutional neural network architecture (image source: Phung and Rhee (2018))

# Appendix D

# Optical flow and learning-based methods for optical flow estimation from events

This chapter will first introduce the concept of optical flow in Section D.1. Then it will proceed to describe a variety of methods of determination of optical flow from event stream using the machine learning paradigm in Sections D.2, D.3 and D.4.

## D.1 Principles of optical flow

Taking the definition from Horn and Schunck (1981) optical flow is *"the distribution of apparent velocities of movement of brightness patterns in an image"*. A more intuitive explanation is presented by Gibson and Marques (2016) who describe optical flow as a projection of motion from a three-dimensional scene into a two-dimensional image plane. In many living organisms including humans, processing optical flow is critical for steering their motion as pointed out in Warren et al. (2001). Accurate determination of optical flow also has a huge importance for robotics including navigation during planetary landing. This section will briefly introduce optical flow and motion field. It will also mention an important issue in estimation of optical flow called aperture problem.

### D.1.1 Optical flow and motion field

It is extremely important to make a correct distinction between optical flow and the so-called motion field. As Jahne et al. (1999) explain, motion field is an actual projection of the three-dimensional velocities on the two-dimensional image plane. On other hand optical flow is really just a distribution of *apparent* velocities of brightness patterns. According to Distante and Distante (2020), non-zero optical flow is a result of either

- Motion of the camera
- Motion in the scene
- Change in lighting

There are cases where there is a motion field, but no optical flow and vice versa. Jahne et al. (1999) bring an excellent example with a uniform-colored sphere and a point lighting source which is shown in the Figure D.1. If the sphere rotates, but the lighting stays constant the brightness patterns will not change and there will be zero optical flow and a non-zero motion field. On the other hand if the lighting source moves, but the sphere is fixed, there will be no motion field, but a non-zero optical flow.

Furthermore, a variant exists where there is both non-zero optical flow and motion field, but they are different. Very famous example brought up by Gibson and Marques (2016) is the barber's pole shown in the Figure D.2. If an observer focuses on one of the stripes when the pole rotates around its axis, they will see it moving down along the axis of the pole. Motion field and optical flow are then orthogonal to each other. This is a manifestation of an effect called aperture problem which is a topic of the next subsection.

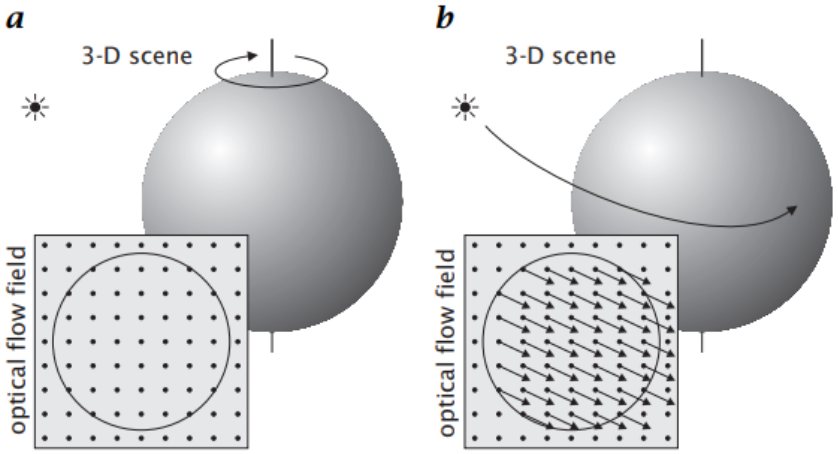



Figure D.1: Demonstration of difference between optical flow and motion field. Case a, that is rotating sphere and fixed lighting source will result in a non-zero motion field, but a zero optical flow. On the other hand case b, fixed sphere and moving lighting source will result in a zero motion field, but a non-zero optical flow. (Image source: Jahne et al. (1999))

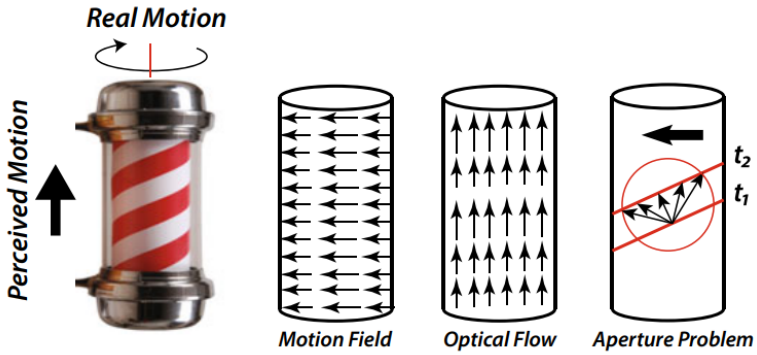



Figure D.2: The barbers pole illusion illustration: while the viewer perceives apparent vertical motion in fact the motion field is horizontal. This ambiguity is caused by so-called aperture problem which prevents the viewer from estimating the exact direction of motion of a straight edge. (Image source: Distante and Distante (2020))

### D.1.2 Aperture problem

When estimating optical flow, as Gibson and Marques (2016) state, a very common assumption is that a point moving in the three-dimensional scene will appear in the projection with a constant brightness $I$ over a short time span. This is called the brightness constancy assumption and without it, it is almost impossible to calculate the optical flow. As Jahne et al. (1999) state, with an assumption of very small motion this leads to a following differential equation with optical flow $\hat{\mathbf{u}}(\hat{x}, \hat{y}, t)$

$$\nabla I(\hat{x}, \hat{y}, t)^T \cdot \hat{\boldsymbol{u}}(\hat{x}, \hat{y}, t) + \frac{\partial I(\hat{x}, \hat{y}, t)}{\partial t} = 0 \tag{D.1}$$

Careful study reveals that this equation contains two unknown variables as the optical flow vector $\hat{\boldsymbol{u}}(\hat{x}, \hat{y}, t)$ is two-dimensional and thus is under-constrained. However because of the properties of dot product it is possible to reconstruct the optical flow component $\hat{u}(\hat{x}, \hat{y}, t)_{\parallel}$ parallel to the brightness gradient. According to Gibson and Marques (2016), this phenomenon is known as aperture problem, because as a result if a moving straight edge is viewed through an aperture, it is impossible to calculate the component of motion perpendicular to the edge (and thus parallel to the brightness gradient). This is illustrated by the Figure D.3. However as Jahne et al. (1999) explain, if a corner appears in the aperture it is possible to calculate full optical flow as there are already two independent brightness constancy equations and thus they can be solved as a system.

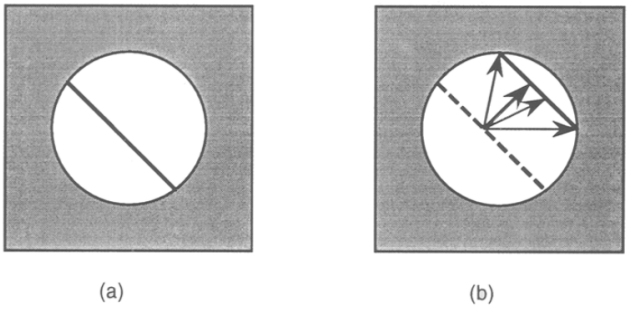



Figure D.3: Illustration of the aperture problem. If a moving edge is observed through an aperture it is possible to determine only the component of motion perpendicular to the intensity gradient. The tangential component can have any magnitude. (Image source: Fermüller (1995))

## D.2 Spiking neural networks using synaptic delays

One of the possible mechanisms to estimate optical flow is by use of synaptic delays. The properly timed delays can be used to make signals from multiple neurons coincide and raise the membrane potential over the critical threshold. This section will discuss use of such strategies.

### D.2.1 Spiking Architecture for Visual Motion Estimation

The first use of spiking neural networks for optical flow estimation can be attributed to Orchard et al. (2013) who developed the Spiking Architecture for Visual Motion Estimation (SAVME). The neurons in SAVME act to recognize pre-defined spatio-temporal event patterns corresponding to an edge moving in certain directions by tuning the synaptic delays of the leaky-integrate and fire neurons.

Each neuron in SAVME sensitive to one speed and one direction and connects to a 5x5 pixel square. An edge moving at approximately the right speed and direction across these pixels will trigger the events in such a manner that they will arrive at the neuron almost at the same time due to the synaptic delays. This will indeed force the membrane voltage over the threshold and the neuron will fire. This principle is illustrated in the Figure D.4.

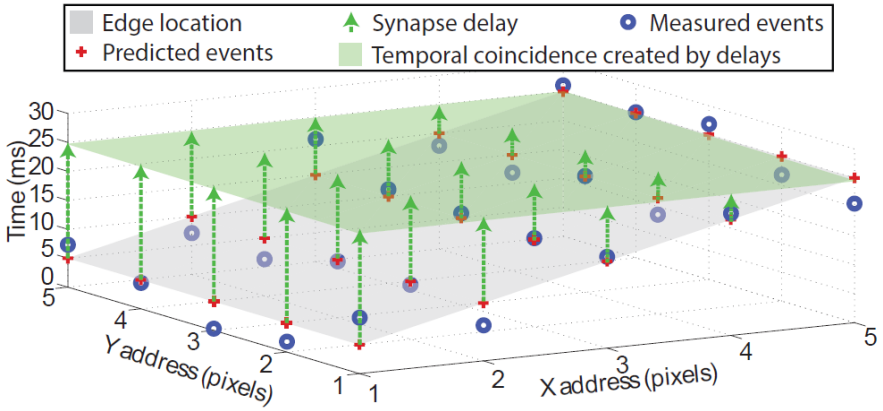



Figure D.4: An illustration of SAVME principle on snapshot of a spatio-temporal space with a moving edge (light grey plane). The edge moves and triggers events. In order to create a temporal coincidence of events on a single neuron (light green plane) the synaptic delays (green arrows) must be tuned. (Image source: Orchard et al. (2013)).

However such firing is still susceptible to the aperture problem. If a straight edge appears in the view, the neuron has no chance of recognizing the flow parallel to the intensity gradient. Solution to this is to add another layer of neurons to incorporate pooling. The neurons can be pooled in such a way that it becomes probable that multiple edges of various orientations fall in their receptive fields. Each pooling neuron is connected to all neurons in the first layer with their specific combination of speed and direction. If then an assumption of local optical flow constancy over the pooled receptive field is made, it is possible to infer the local flow speed and

direction from the specific combination of pooling neurons which fired.

While SAVME is arguably impractical for any larger scale implementations as it requires hand-tuning to specific datasets, it serves as a good proof that spiking neural networks can indeed process event-based optical flow. Moreover it can be implemented even on simple neuromorphic hardware.

### D.2.2 Convolutional neural networks with delay blocks

Another solution utilizing synaptic delays was described by Chaney et al. (2021). A convolutional network is created with four layers utilizing leaky-integrate and fire neurons. The first two layers extract spatial features from the network input followed by a layer of delay blocks. Finally another convolution layer follows which decodes the features. The architecture can be seen in the Figure D.5.

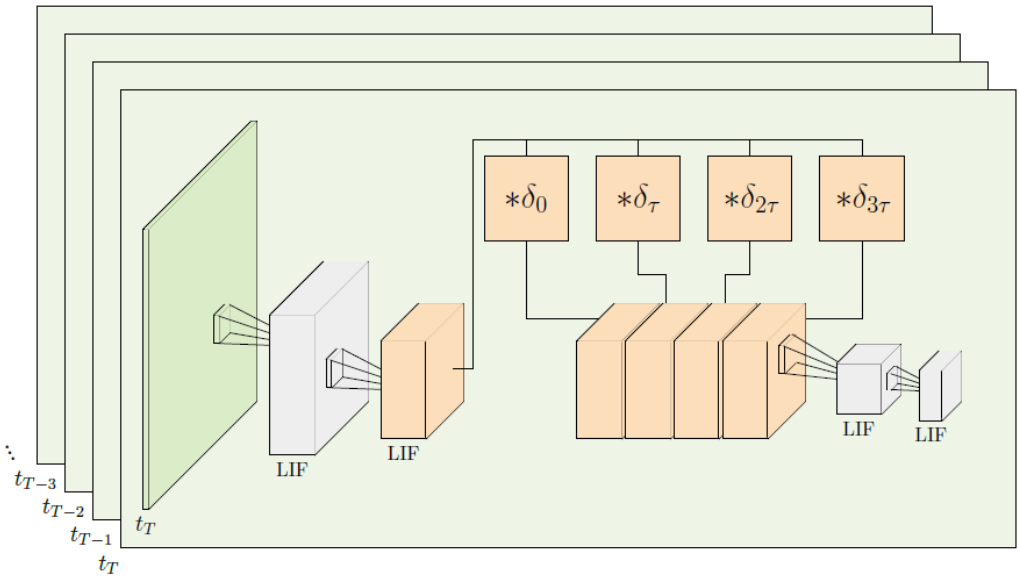



Figure D.5: Architecture of network proposed by Chaney et al. (2021). Two convolution layers are followed by a layer of synaptic delay blocks. The blocks than link directly into another convolution layer. Events are given to the network as a sequence of frames (here marked as $t_{T_i}$). (Image source: Chaney et al. (2021)).

The events are inputted to the network in the form of two-dimensional frames which indicate in binary representation whether an event happened or did not happen on a specific pixel during a very short time window. These frames are fed to the network sequentially.

The delay blocks allow the network to store information between individual frames and explicitly compare the signals. By the use of synaptic delays the network can make use of the temporal context in which individual features appear. The network is using self-supervised learning with a combination of photometric and smoothness loss further described in the Section D.3.1.

## D.3 U-Net-like convolutional neural networks

The by far most common network architecture found in event-based optical flow estimations is the famous U-Net developed by Ronneberger et al. (2015). It is a convolutional neural network originally developed for segmentation of medical images. However it found its way into optical flow estimations which are in a way comparable to segmentation in requiring an input and output of similar dimensions.

As stated in Ronneberger et al. (2015), U-Net consists of contracting path (called encoder) and expansive path (called decoder). In the encoder convolutions and pooling operations downsample the input while the steps of the decoder provide upsampling operations until desired dimensionality is reached. According to Zhou et al. (2020), the success of the U-Net is largely due to clever employment of the so-called skip connections. As Zhou et al. (2020) states the skip connections provide a link between the low-level, fine feature maps in the encoder

to the rather deep feature maps in the decoder.

### D.3.1 Analog U-Nets

Use of U-Net for optical flow from events estimation was pioneered by Zhu, Yuan, et al. (2018) with an analog neural network called the EV-FlowNet. Great results achieved by this implementation made it a popular benchmark and inspiration for most of the other following solutions.

EV-FlowNet makes use of well developed architectures for image processing by presenting the events to the network in an image-like form. Input is a four-channel image with a resolution identical to the one of the camera. The first two channels define a number of positive and negative events on each pixel during a short predefined time window. The other two channels represent the timestamp of last positive and negative event on each pixel.

The network itself consists of four encoder layers with strided convolution followed by a bottleneck of two residual blocks and a decoder which has four layers of upsample convolution. The whole architecture can be seen in the Figure D.6.

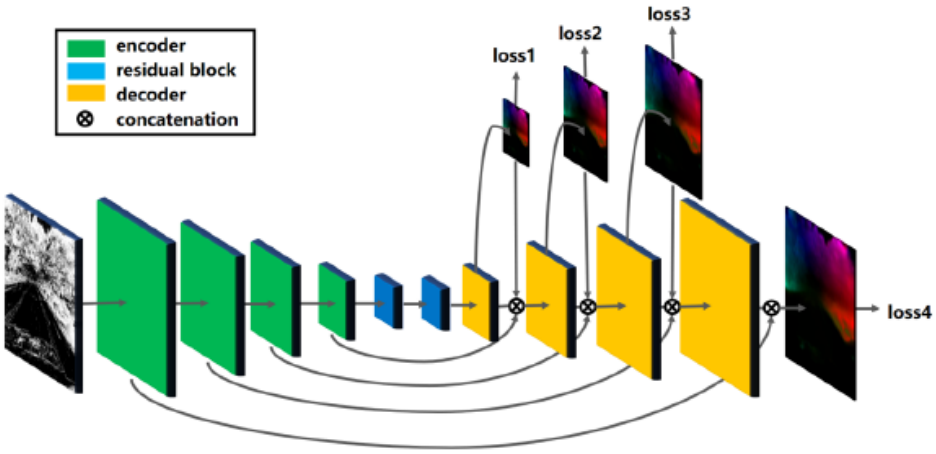



Figure D.6: Architecture of the EV-FlowNet, the encoder layers with strided convolution are followed with two residual blocks and four decoder layers with the upsampling convolution. (Image source: Zhu, Yuan, et al. (2018))

Of note is that the network was trained on datasets from a DAVIS camera (described in the A.3, so both events and grayscale images are available. This allowed the authors to do self-supervised training using a so-called photometric loss function where an image is taken at the beginning and at the end of a time window where events are gathered, the first image is then warped according to the resultant optic flow and the loss is calculated based on a difference in intensity between the warped image and the second image. Zhu, Yuan, et al. (2018) however note that to deal with the aperture problem it is necessary to also add smoothness term to the loss function which aims to minimize the difference in optical flow between neighboring pixels.

The authors of EV-FlowNet realized that summing up the events into a histogram and keeping only times of last events for each pixel leads to discarding important information. Therefore in a follow up work by Zhu, Yuan, et al. (2018) the events are distributed into a three-dimensional voxel grid (with time as the third dimension) using bilinear sampling and this grid acts as an input to the network. It is noted that the voxels contain binary information and there is no way to find whether more than one event is connected to a voxel. This novel input method leads to an increase in performance over the EV-FlowNet.

Sun et al. (2022) also propose a novel way of representing event data in EV-FlowNet similar to Zhu, Yuan, et al. (2018). In the work the input is divided into four channels based on event polarity and belonging to either half of the temporal window. Authors show reduction in computational expense in comparison to previous methods.

### D.3.2 Spiking and analog hybrid U-Nets

Spike-FlowNet by C. Lee et al. (2020) is a U-Net-like hybrid between a spiking and analog neural network used to estimate optical flow. As noted by S. Lee and Kim (2021), deep spiking neural networks have performance issues due to a phenomenon called spike vanishing. On the other hand analog neural networks are incapable of utilizing the precise timing information and show less computational efficiency than spiking neural networks which are well-suited for processing the asynchronous event stream. Spike-FlowNet therefore uses spiking neural network for processing inputs and analog neural network for reconstruction of the output.

The architecture of the network is de-facto identical to the EV-FlowNet (described in the Section D.3.1) with the difference that the entire encoder is implemented as a spiking neural network with integrate and fire type neurons as can be seen in the D.7. Also similarly to EV-FlowNet the network is trained using self-supervised learning with a combination of smoothness and photometric loss.

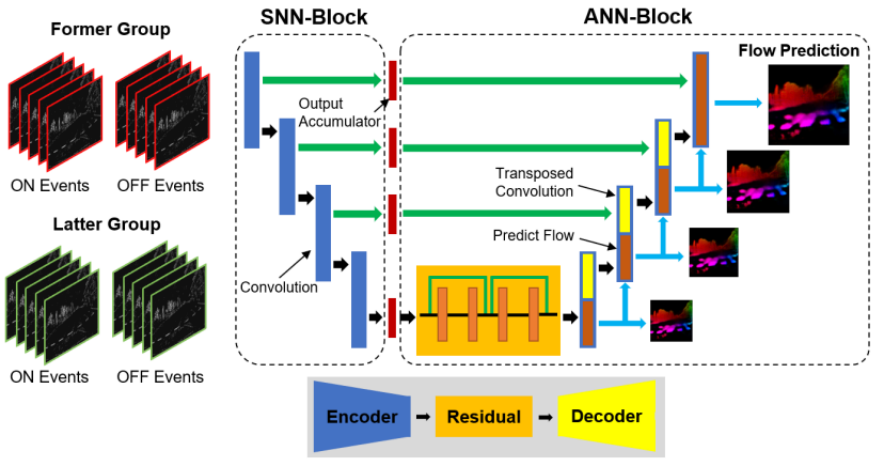



Figure D.7: An illustration of Spike-FlowNet architecture. The four channels are sequentially fed into the spiking encoder followed by the analog decoder. (Image source: C. Lee et al. (2020))

The key difference to the EV-FlowNet is however the handling of the inputs. Instead of summing all the events over a temporal window, the spiking encoder allows for the window to be sliced into very short temporal frames which are then passed sequentially to the network. According to C. Lee et al. (2020), this eliminates the need for additional channels for timestamps and naturally preserves the temporal context for all events.

Training a hybrid network is more complex than training a pure analog network.

The Spike-FlowNet proved to be very successful and according to C. Lee et al. (2020) at the time of its release managed to outperform the state-of-the-art analog neural networks trained for the same purpose. On the basis of Spike-FlowNet, C. Lee et al. (2021) proposed Fusion-FlowNet which is combines event and frame-based data in order to generate dense estimates of optical flow rather than limiting the estimate only to pixels where events happened during the relevant time window.

Recently a work of Yang et al. (2023) emerged which presents a system similar to Spike-FlowNet, but adds several novel concepts. Most importantly attention mechanisms were added to all outputs of the encoder. As explained by M.-H. Guo et al. (2022), attention mechanisms allow for easy prioritization of certain more important features being extracted by the encoder over the less important ones. Yang et al. (2023) note that the attention mechanisms brought an increase in performance.

### D.3.3 Fully spiking U-Nets

Despite the problems with the vanishing spikes in deep spiking neural networks mentioned by C. Lee et al. (2020) there have been several fully spiking U-Nets implemented.

Among the first ones was Hagenaars et al. (2021) presenting SNN-EV-FlowNet which is a translation of EV-FlowNet architecture (for details please refer to the Section D.3.1) to a spiking neural network with leaky-integrate and fire neurons. As noted by Hagenaars et al. (2021) the solution did suffer from spike vanishing issues, but yet managed to produce competitive results. Interesting thing about the SNN-EV-FlowNet is a use

of contrast maximization for a loss function for self-supervised learning. Training is done using the surrogate gradient method.

Important is work of Kosta and Roy (2023) who remade and tuned the Spike-FlowNet (described in the Section D.3.2) and managed to produce a fully spiking version called Adaptive-SpikeNet trained using surrogate gradients. Their work currently produces state-of-the-art results.

A major topic in implementation of spiking neural networks is the compatibility with the common neuromorphic hardware (further elaborated in the Section C.2.5). SteroSpike of Rançon et al. (2021) is a U-Net-like depth estimation network rather than one for optical flow, but it shown important paths forward to achieve hardware-friendly solutions. Most of the parts of the network are common to e.g. Kosta and Roy (2023), but the decoder uses nearest neighbor upsampling which does not use floating point operation unlike the works of Hagenaars et al. (2021) or Kosta and Roy (2023). Moreover the authors of StereoSpike were concerned with lack of expressivity in such networks and introduced an architecture where predictions from each stage of the decoder from coarse to fine contribute equally to the final prediction. Such a strategy increases the number of spikes which can influence the readout neurons and improves performance.

Work of Cuadrado et al. (2023) came greatly inspired by the StereoSpike. Their optical flow estimation U-Net utilizes both the nearest neighbor upsampling and linking or the readout neurons to deep decoder layers. However there are more novel approaches presented by the authors.

Firstly the traditional spatial convolution is replaced by maximum pooling in Cuadrado et al. (2023). This improves performance and is found to be neuromorphic hardware-friendly. Secondly the input to the network are 21 event frames with two channels (negative and positive events) concatenated along temporal direction. Use is then made of temporal convolution to extract temporal features in the encoder. The temporal convolution is done in such a way that the temporal dimension is flattened before the bottleneck and in the decoder only two dimensions are considered. This dictates that only the last temporal event frame in each encoder layer can be used in the skip connections. The whole architecture is visible in the Figure D.8.

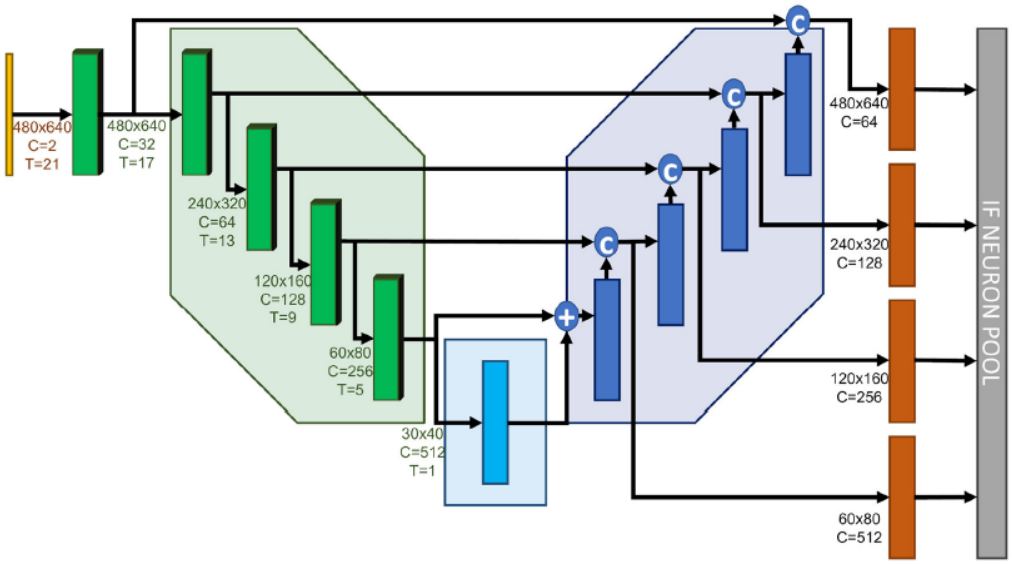



Figure D.8: An illustration architecture used by Cuadrado et al. (2023). Note the decreasing temporal dimension in the encoder (here marked as T) due to the temporal convolution. Also note how all the decoder layers are linked to the readout neurons. (Image source: Cuadrado et al. (2023))

Schnider et al. (2023) introduce the use of a novel Timelens architecture which is rather similar to U-Net. It uses leaky integrate and fire neurons with a rather interesting trainable recurrent parameter which helps the neurons to better capture temporal dependencies.

## D.4 Other spiking architectures

Notable is work of Paredes-Vallés et al. (2018) devising a fully spiking convolutional network for egomotion estimation. The network consists of three convolution layers, one pooling layer and a dense layer at the output.

It is important to realize that the prediction is the egomotion not the optical flow, so unlike for the other cases described in this chapter, the input and output have a vastly different dimensionality.

Probably the most interesting feature of the work of Paredes-Vallés et al. (2018) is the self-supervised learning rule utilizing spike-timing-dependent plasticity. This strategy formalizes the idea that neurons which fire together have strong connections presented by Hebb (1949). A neural network rather similar to the one of Paredes-Vallés et al. (2018) was also proposed by Barbier et al. (2021), but specifically tuned for binocular cameras and aiming at also estimating orientation and binocular disparity next to motion. The architecture of Paredes-Vallés et al. (2018) is also adopted by Tian and Andrade-Cetto (2023) in a novel method to estimate egomotion from a noisy optical flow.

Hagenaars et al. (2021) proposed a U-Net-like network described in the Subsection D.3.3, but also a lightweight network called SNN-FireNet based on the FireNet proposed by Scheerlinck et al. (2020) for fast image reconstruction.

The SNN-FireNet architecture can be seen in the Figure D.9. It is notable that it utilizes Gated Recurrent Units which as explained by Scheerlinck et al. (2020) allow the network to capture historical information into a hidden state which allows to integrate temporal information into the predictions. The SNN-FireNet was found to produce results rather similar to the U-Net-like network and encountered issues with spike vanishing.

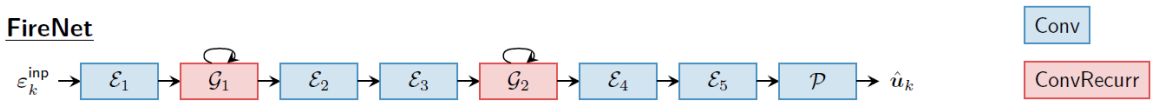



Figure D.9: SNN-FireNet architecture used by Hagenaars et al. (2021). Note the use of Gated Recurrent units. Image taken and modified from Hagenaars et al. (2021).

In Y. Zhang et al. (2023) a novel way of inputting events to a spiking neural network can be found by using adaptive algorithm to slice the event frames in a most beneficial way such that they show clear and sharp edges. An intuition can be made that faster movement forces the algorithm to do faster slicing and vice versa. The details of the method can be found in Y. Zhang et al. (2022). Apart from that Y. Zhang et al. (2023) introduce a simple encoder/decoder architecture with leaky-integrate and fire neurons.

## D.5 Other analog architectures

This section will briefly show some of the analog neural networks which cannot be classified as U-Nets. While they may not be directly related to the topic of the work, they still contain very valid and interesting ideas.

M. Gehrig, Millhäusler, et al. (2021) developed E-RAFT which similarly to e.g. Hagenaars et al. (2021) uses Gated Recurrent Units to capture the state data and make use of temporal dependencies between the spikes in a convolutional neural network. The recurrence is exploited by the use of the so-called cost volumes which are well-suited for comparison of the different event frames. Work of Ding et al. (2021) can be compared to M. Gehrig, Millhäusler, et al. (2021) as it builds on very similar principles.

An approach very different from all other approaches in this chapter was taken by Nagata and Aoki (2022). The spatio-temporal plane fitting strategies, which are common model-based approach to event-based optical flow, suffer greatly from noise when the plane is being fitted with least-squares minimization. Nagata and Aoki (2022) utilize the so-called PointNet neural network architecture which is very well suited to find the outliers caused by noise effects in the event clouds and improve the plane fitting. This is done in a very computationally efficient manner. The efficient outlier detection is deemed interesting even in context of combination with other methods.

# Appendix E

# Additional results

This appendix contains additional results on updated dataset obtained at the end of the research, which could not be fully analyzed in the available timespan, yet they are considered potentially interesting to the reader.

## E.1 Dataset update

It was found that by fine-tuning PANGU level-of-detail settings and generating imagery at higher than necessary resolution, the popping effect described in Subsection 4.2.2 can be essentially suppressed. This in turn allows to increase the frames rate of the source videos to values around 1000 frames per second without significant undesirable artifacts. As such, there is no need to fear discrete effects of the v2e simulator described in Subsection 4.2.2. The entire dataset was regenerated using this new method. Figure E.1 illustrates this difference by showing brightness gradients present in the original and updated source imagery, where the level of detail steps are no longer visible.

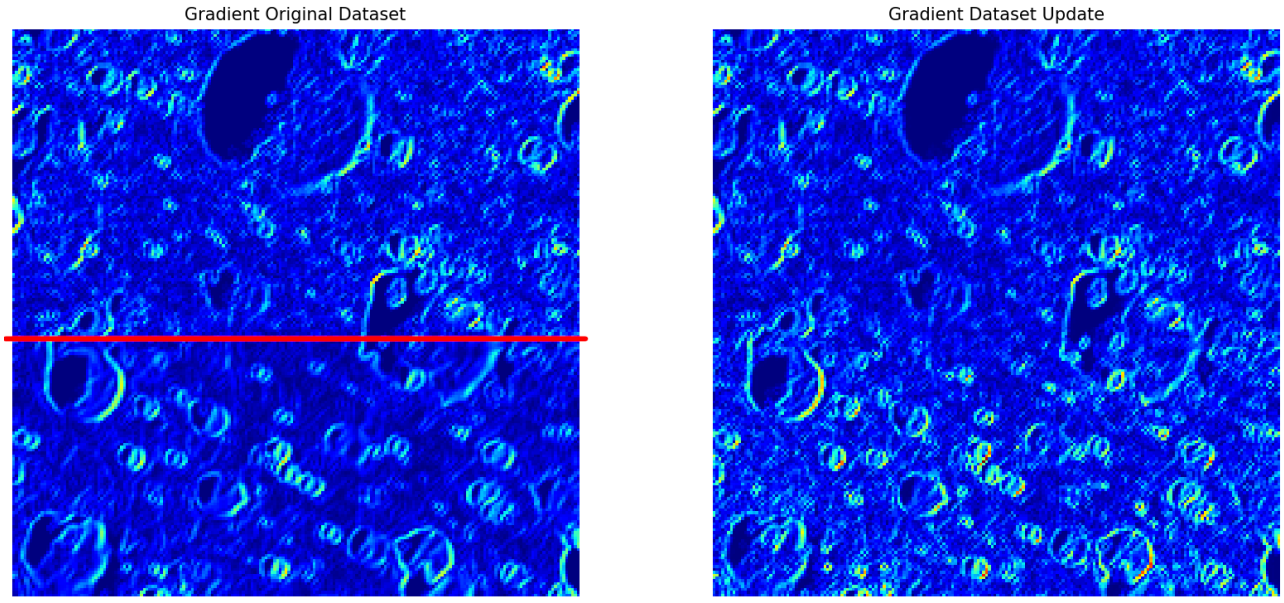



Figure E.1: Comparison between the original dataset and its update. Brightness spatial gradient magnitude is plotted in the image to highlight the level of detail. Notice that above the red line both of the images are the same, but below the red line there is a lower level of detail present in the image from original dataset.

## E.2 Model retraining

Due to time constraints only two models could be retrained on the new dataset. To support the conclusions of RSQ-2, it was decided to retrain ASN AC and OES AC. Table E.1 presents the AEE values achieved on the updated dataset. It can be seen that, compared to the original data set, both models slightly improve in performance. Interestingly, performance on nominal and divert trajectories is now very similar for both models,

whereas previously the nominal trajectories were clearly an easier task. The original results are however not considered to be invalidated by this update.

| Model | V | N | D | Sum |
|---|---|---|---|---|
| ASN AC | 1.02 | 3.28 | 3.15 | 7.45 |
| OES AC | 1.12 | 2.12 | 2.14 | 5.38 |

Table E.1: Mean AEE values for models across trajectory types (V, N, D) and their totals on the improved dataset

Visual investigation revealed an interesting tendency in ASN AC to reach very large angular errors of the optical flow vectors during fast attitude maneuvers to an extent unseen in the original dataset. On the other hand, OES AC seems to be more capable of handling such conditions than it was before the update. Figure E.2 presents a plot of average angular error of optical flow against angular velocity.

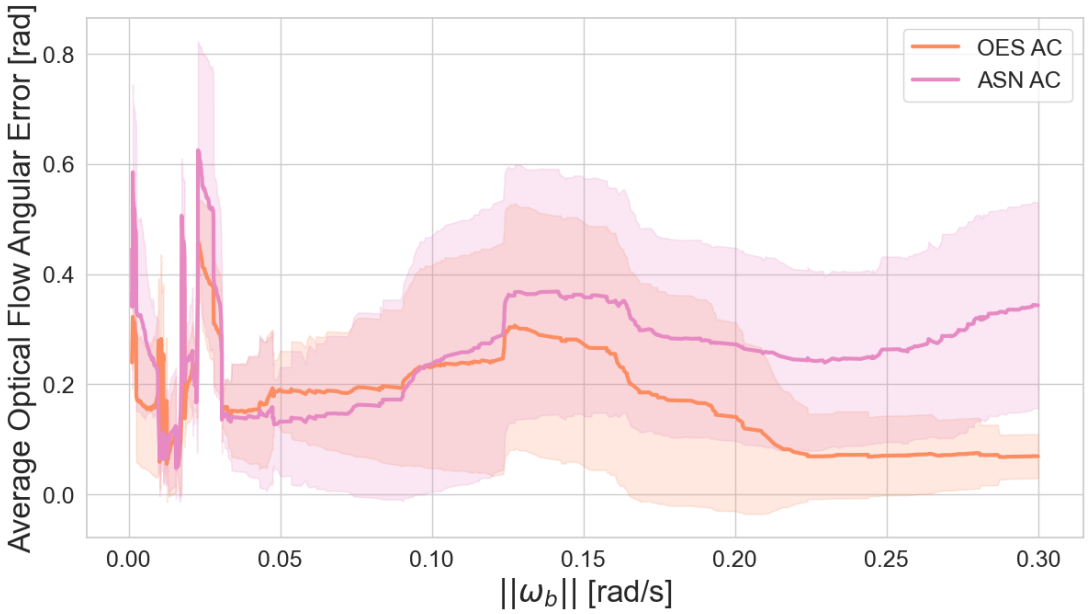



Figure E.2: Plot of average angular error in the optical flow against the angular velocity in updated ASN AC and OES AC models. Notice that the performance of ASN AC significantly deteriorates at higher angular velocities and OES AC feels exactly the opposite effect. Shaded area corresponds to one standard deviation.

This prompted a search for possible reasons behind this behavior. Two hypotheses were posed. Firstly, it is possible that OES AC is simply better than ASN AC at training for regimes, which are only sparsely seen in the dataset and requires less data. Second option is that ASN AC puts a large emphasis on the event count in different parts of the image plane, which however becomes approximately uniform during fast attitude maneuvers. The second hypothesis however, could not be confirmed by studies using localized large-scale event dropout as both models proved to be rather insensitive to this type of test.

To follow up on Chapter 7, ACEE was measured on the same trajectories using the updated OES AC model. The results are presented in Table E.2. Table E.3 then provides a comparison with results from Chapter 7 before the update. The results seem to be rather similar, with the exception that the new model proves to perform very significantly worse on nominal and ventral trajectories in KAA regime and it is markedly better on divert trajectories. Reasons for this behaviour are not yet known and they can be further studied in the future.

| Regime/Trajectory type | V | N | D |
|---|---|---|---|
| KAA | 0.015 ($\sigma = 0.005$) | 0.018 ($\sigma = 0.010$) | 0.022 ($\sigma = 0.011$) |
| R | 0.016 ($\sigma = 0.005$) | 0.019 ($\sigma = 0.010$) | 0.038 ($\sigma = 0.019$) |
| KA | 0.017 ($\sigma = 0.005$) | 0.026 ($\sigma = 0.014$) | 0.044 ($\sigma = 0.019$) |

Table E.2: Updated ACEE values using the OES AC model trained on the improved dataset.

| Regime/Trajectory type | V | N | D |
|---|---|---|---|
| **KAA** | +25% | +80% | -4% |
| **R** | -6% | 0% | -5% |
| **KA** | 0% | +8% | -8% |

Table E.3: ACEE difference between OES AC models before and after update in percent. Higher values indicate poorer performance of the new model relative to the old model and vice versa.

# Bibliography


Açıkmeşe, B., Casoliva, J., Carson, J., & Blackmore, L. (2012). G-fold: A real-time implementable fuel optimal large divert guidance algorithm for planetary pinpoint landing. *LPI Contributions*, 4193–.

Açıkmeşe, B., & Polen, S. (2007). Convex programming approach to powered descent for mars landing. *J. Guidance Control Dyn.*, *44*, 310–322.

Aggarwal, C. C. (2018). *Neural networks and deep learning*. Springer. https://doi.org/10.1007/978-3-319-94463-0

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.-J., Taba, B., Beakes, M., Brezzo, B., Kuang, J., Manohar, R., Risk, W., Jackson, B., & Modha, D. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *34*, 1537–1557. https://doi.org/10.1109/TCAD.2015.2474396

Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., & Diehl, M. (2019). CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, *11*(1), 1–36. https://doi.org/10.1007/s12532-018-0139-4

Armenise, M., Ciminelli, C., Dell'Olio, F., & Passaro, V. (2011). *Advances in gyroscope technologies*. https://doi.org/10.1007/978-3-642-15494-2

Avvenire, A. T. (1974). *Apollo 16, lm-11 descent propulsion system final flight evaluation* (tech. rep. No. NASA-CR-19740024177). NASA.

Azzalini, L. J., Blazquez, E., Hadjiivanov, A., Meoni, G., & Izzo, D. (2023). On the generation of a synthetic event-based vision dataset for navigation and landing.

Bains, S. (2021). Benchmarking neuromorphic computing: Devil is in the details. https://www.eetimes.eu/benchmarking-neuromorphic-computing-devil-is-in-the-details/

Ball, A., Garry, J., Lorenz, R., & Kerzhanovich, V. (2007). *Planetary landers and entry probes*. Cambridge University Press.

Barbier, T., Teuliere, C., & Triesch, J. (2021). Spike timing-based unsupervised learning of orientation, disparity, and motion representations in a spiking neural network. https://doi.org/10.1109/CVPRW53098.2021.00152

Bartlett, W., Kirkland, Z. D., Polifka, R. W., Smithson, J. C., & Spencer, G. L. (1966). *Design specifications for apollo spacecraft liquid primary propulsion systems* (tech. rep. No. NASA-TM-X-58040). NASA.

Bennett, F. V. (1970). *Apollo lunar descent and ascent trajectories* (tech. rep. No. NASA-TM-X-58040). NASA.

Betts, J. T. (2010). *Practical methods for optimal control and estimation using nonlinear programming, second edition*. Society for Industrial; Applied Mathematics. https://doi.org/10.1137/1.9780898718577

Blackmore, L. (2016). Autonomous precision landing of space rockets. *46*, 15–20.

Bohté, S. M., Kok, J. N., & Poutré, H. L. (2000). Spikeprop: Backpropagation for networks of spiking neurons. *The European Symposium on Artificial Neural Networks*. https://api.semanticscholar.org/CorpusID:14069916

Bonalli, R., Cauligi, A., Bylard, A., & Pavone, M. (2019). Gusto: Guaranteed sequential trajectory optimization via sequential convex programming.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Brandli, C., Berner, R., Yang, M., Liu, S.-C., & Delbruck, T. (2014). A 240 × 180 130 db 3 µs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, *49*(10), 2333–2341. https://doi.org/10.1109/JSSC.2014.2342715

Brochard, R., Lebreton, J., Robin, C., Kanani, K., Jonniaux, G., Masson, A., Despré, N., & Berjaoui, A. (2018). Scientific image rendering for space scenes with the surrender software.

Brunel, N., & van Rossum, M. (2008). Lapicque's 1907 paper: From frogs to integrate-and-fire. *Biological cybernetics*, *97*, 337–9. https://doi.org/10.1007/s00422-007-0190-0

Chaney, K., Panagopoulou, A., Lee, C., Roy, K., & Daniilidis, K. (2021). Self-supervised optical flow with spiking neural networks and event based cameras. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5892–5899. https://doi.org/10.1109/IROS51168.2021.9635975

Chu, H., Ma, L., Wang, K., Shao, Z., & Song, Z. (2017). Trajectory optimization for lunar soft landing with complex constraints. *Advances in Space Research*, *60*. https://doi.org/10.1016/j.asr.2017.07.024

Conway, B. (2010). The problem of spacecraft trajectory optimization. *Spacecraft Trajectory Optimization*, 1–15. https://doi.org/10.1017/CBO9780511778025.002

Cuadrado, J., Rançon, U., Cottereau, B. R., Barranco, F., & Masquelier, T. (2023). Optical flow estimation from event-based cameras and spiking neural networks. *Frontiers in Neuroscience*, *17*. https://doi.org/10.3389/fnins.2023.1160034

Date, P., Kay, B., Schuman, C., Patton, R., & Potok, T. (2021). Computational complexity of neuromorphic algorithms. https://doi.org/10.1145/3477145.3477154

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.-K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., ... Wang, H. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99. https://doi.org/10.1109/MM.2018.112130359

Diamond, S., & Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, *17*(83), 1–5.

Diehl, P. U., Zarrella, G., Cassidy, A. S., Pedroni, B. U., & Neftci, E. (2016). Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. *CoRR*, *abs/1601.04187*. http://arxiv.org/abs/1601.04187

Ding, Z., Zhao, R., Zhang, J., Gao, T., Xiong, R., Yu, Z., & Huang, T. (2021). Spatio-temporal recurrent networks for event-based optical flow estimation. *CoRR*, *abs/2109.04871*. https://arxiv.org/abs/2109.04871

Distante, A., & Distante, C. (2020). Motion analysis. In *Handbook of image processing and computer vision: Volume 3: From pattern to object* (pp. 479–598). Springer International Publishing. https://doi.org/10.1007/978-3-030-42378-0_6

Domahidi, A., Chu, E., & Boyd, S. (2013). Ecos: An socp solver for embedded systems. *2013 European Control Conference (ECC)*, 3071–3076. https://doi.org/10.23919/ECC.2013.6669541

Duchi, J. (2018). Sequential convex programming: Notes for ee364b, stanford university.

Eshraghian, J. K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., & Lu, W. D. (2021). Training spiking neural networks using lessons from deep learning. *CoRR*, *abs/2109.12894*. https://arxiv.org/abs/2109.12894

Fang, W., Chen, Y., Ding, J., Yu, Z., Masquelier, T., Chen, D., Huang, L., Zhou, H., Li, G., & Tian, Y. (2023). Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence. https://arxiv.org/abs/2310.16620

Fermüller, C. (1995). Passive navigation as a pattern recognition problem. *International Journal of Computer Vision*, *14*, 147–158. https://doi.org/10.1007/BF01418980

Fourer, R. (1996). *Ampl : A modeling language for mathematical programming*. San Francisco, Calif. : Scientific Pr.

Foust, J. (2024). Intuitive machines and nasa call im-1 lunar lander a success as mission winds down.

Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., Leutenegger, S., Davison, A., Conradt, J., Daniilidis, K., & Scaramuzza, D. (2019). Event-based vision: A survey.

Gehrig, D., Gehrig, M., Hidalgo-Carrió, J., & Scaramuzza, D. (2020). Video to events: Recycling video datasets for event cameras.

Gehrig, D., Rebecq, H., Gallego, G., & Scaramuzza, D. (2018). Asynchronous, photometric feature tracking using events and frames. https://doi.org/10.1007/978-3-030-01258-8_46

Gehrig, M., Aarents, W., Gehrig, D., & Scaramuzza, D. (2021). Dsec: A stereo event camera dataset for driving scenarios. https://arxiv.org/abs/2103.06011

Gehrig, M., Millhäusler, M., Gehrig, D., & Scaramuzza, D. (2021). E-raft: Dense optical flow from event cameras.

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge University Press.

Gibson, J., & Marques, O. (2016). *Optical flow and trajectory estimation methods*. Springer.

Gill, P. E., Murray, W., & Saunders, M. A. (2005). Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Rev.*, *47*(1), 99–131. https://doi.org/10.1137/S0036144504446096

Grabe, V., Bülthoff, H., Scaramuzza, D., & Giordano, P. (2015). Nonlinear ego-motion estimation from optical flow for online control of a quadrotor uav. *The International Journal of Robotics Research*, *34*. https://doi.org/10.1177/0278364915578646

Guo, M.-H., Xu, T.-X., Liu, J.-J., Liu, Z.-N., Jiang, P.-T., Mu, T.-J., Zhang, S.-H., Martin, R. R., Cheng, M.-M., & Hu, S.-M. (2022). Attention mechanisms in computer vision: A survey. *Computational Visual Media*, *8*(3), 331–368. https://doi.org/10.1007/s41095-022-0271-y

Guo, W., Fouda, M. E., Eltawil, A. M., & Salama, K. N. (2021). Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. *Frontiers in Neuroscience*, *15*. https://doi.org/10.3389/fnins.2021.638474

Hagenaars, J., & Paredes-Vallés, F. (2023). tudelft/spiking.

Hagenaars, J., Paredes-Vallés, F., & de Croon, G. (2021). Self-supervised learning of event-based optical flow with spiking neural networks.

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Wiley.

Henley, C. (2021). *Foundations of neuroscience*. Michigan State University Libraries.

Hirasawa, R., Hashimoto, T., Tokunaga, K., Nakajima, S., Miyoshi, K., Hirose, C., Kikuchi, J., Bando, N., Morishita, N., Tomiki, A., Torii, W., Ito, T., Otsuki, M., Yoshimitsu, T., Ishige, Y., Takeuchi, H., & Yamamoto, Y. (2025). Operational plan, results and recovery scenarios of cubesat lunar lander omotenashi. In Y. H. Lee, A. Schmidt, & E. Trollope (Eds.), *Space operations* (pp. 695–720). Springer Nature Switzerland.

Hodgkin, A., & Huxley, A. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, *117*, 500–544.

Hordijk, B., Scheper, K., & Croon, G. (2018). Vertical landing for micro air vehicles using event-based optical flow. *Journal of Field Robotics*, *35*, 69–90. https://doi.org/10.1002/rob.21764

Horn, B., & Schunck, B. (1981). Determining optical flow. *Artificial Intelligence*, *17*, 185–203. https://doi.org/10.1016/0004-3702(81)90024-2

Hu, Y., Liu, S., & Delbrück, T. (2020). V2E: from video frames to realistic DVS event camera streams. *CoRR*, *abs/2006.07722*. https://arxiv.org/abs/2006.07722

Ishida, T., Fukuda, S., Kariya, K., Kamata, H., Takadama, K., Kojima, H., Sawai, S., & Sakai, S. (2025). Vision-based navigation and obstacle detection flight results in slim lunar landing. *Acta Astronautica*, *226*, 772–781. https://doi.org/https://doi.org/10.1016/j.actaastro.2024.11.002

Izhikevich, E. (2004). Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, *15*(5), 1063–1070. https://doi.org/10.1109/TNN.2004.832719

Izzo, D., & Croon, G. (2012). Landing with time-to-contact and ventral optic flow estimates. *JOURNAL OF GUIDANCE CONTROL AND DYNAMICS*, *35*, 1362–. https://doi.org/10.2514/1.56598

Jahne, B., Haussecker, H., & Geissler, P. (1999). *Handbook of computer vision and applications: Volume 2: From images to features*. Academic Press, Inc.

Jia, Y., Liu, L., Wang, X., Guo, N., & Wan, G. (2022). Selection of lunar south pole landing site based on constructing and analyzing fuzzy cognitive maps. *Remote Sensing*, *14*(19). https://doi.org/10.3390/rs14194863

Jiang, H., Sun, D., Jampani, V., Yang, M.-H., Learned-Miller, E., & Kautz, J. (2018). Super slomo: High quality estimation of multiple intermediate frames for video interpolation.

Joubert, D., Marcireau, A., Ralph, N. O., Jolley, A., van Schaik, A., & Cohen, G. (2021). Event camera simulator improvements via characterized parameters. *Frontiers in Neuroscience*, *15*. https://api.semanticscholar.org/CorpusID:236437549

Kaiser, J., v. Tieck, J. C., Hubschneider, C., Wolf, P., Weber, M., Hoff, M., Friedrich, A., Wojtasik, K., Roennau, A., Kohlhaas, R., Dillmann, R., & Zöllner, J. (2016). Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks. https://doi.org/10.1109/SIMPAR.2016.7862386

Kim, W.-T., Park, C., Lee, H., Lee, I., & Lee, B.-G. (2019). A high full well capacity cmos image sensor for space applications. *Sensors*, *19*, 1505. https://doi.org/10.3390/s19071505

Klumpp, A. R. (1974). Apollo lunar descent guidance. *Autom.*, *10*, 133–146. https://api.semanticscholar.org/CorpusID:121068650

Kosta, A. K., & Roy, K. (2023). Adaptive-spikenet: Event-based optical flow estimation using spiking neural networks with learnable neuronal dynamics.

Lee, A., Ely, T., Sostaric, R., Strahan, A., Riedel, J., Ingham, M., Wincentsen, J., & Sarani, S. (2010). Preliminary design of the guidance, navigation, and control system of the altair lunar lander. *AIAA Guidance, Navigation, and Control Conference*. https://doi.org/10.2514/6.2010-7717

Lee, C., Kosta, A., Zhu, A. Z., Chaney, K., Daniilidis, K., & Roy, K. (2020). Spike-flownet: Event-based optical flow estimation with energy-efficient hybrid neural networks. *CoRR*, *abs/2003.06696*. https://arxiv.org/abs/2003.06696

Lee, C., Kosta, A. K., & Roy, K. (2021). Fusion-flownet: Energy-efficient optical flow estimation using sensor fusion and deep fused spiking-analog network architectures.

Lee, S., & Kim, H. J. (2021). Low-latency and scene-robust optical flow stream and angular velocity estimation. *IEEE Access*, *9*, 155988–155997. https://doi.org/10.1109/ACCESS.2021.3129256

Leonard, D., & Van Long, N. (1992). *Optimal control theory and static optimization in economics*. Cambridge University Press.

Li, P., Garratt, M., & Lambert, A. (2015). A homography-based visual inertial fusion method for robust sensing of a micro aerial vehicle. https://doi.org/10.1109/ICMA.2015.7237534

Li, S., Jiang, X., & Tao, T. (2015). Guidance summary and assessment of the chang'e-3 powered descent and landing. *Journal of Spacecraft and Rockets*, *53*. https://doi.org/10.2514/1.A33208

Lichtsteiner, P., Posch, C., & Delbruck, T. (2008). A 128× 128 120 db 15 s latency asynchronous temporal contrast vision sensor. *Solid-State Circuits, IEEE Journal of*, *43*, 566–576. https://doi.org/10.1109/JSSC.2007.914337

Lin, S., Ma, Y., Guo, Z., & Wen, B. (2022). Dvs-voltmeter: Stochastic process-based event simulator for dynamic vision sensors. *ECCV*.

Lobo, J. L., Ser, J. D., Bifet, A., & Kasabov, N. K. (2019). Spiking neural networks and online learning: An overview and perspectives. *CoRR*, *abs/1908.08019*. http://arxiv.org/abs/1908.08019

Luenberger, D. G. (2003). *Linear and nonlinear programming*. Kluwer Academic Publishers.

Maass, W., & Bishop, C. (Eds.). (1999). *Pulsed neural networks*. MIT Press.

Maheshwari, S., Serb, A., Papavassiliou, C., & Prodromakis, T. (2022). An adiabatic capacitive artificial neuron with rram-based threshold detection for energy-efficient neuromorphic computing. https://doi.org/10.48550/arXiv.2202.01144

Malyuta, D., Reynolds, T. P., Szmuk, M., Lew, T., Bonalli, R., Pavone, M., & Açıkmeşe, B. (2021). Convex optimization for trajectory generation.

Malyuta, D., Reynolds, T. P., Szmuk, M., Lew, T., Bonalli, R., Pavone, M., & Açıkmeşe, B. (2022). Convex optimization for trajectory generation: A tutorial on generating dynamically feasible trajectories reliably and efficiently. *IEEE Control Systems*, *42*(5), 40–113. https://doi.org/10.1109/mcs.2022.3187542

Malyuta, D., Yu, Y., Elango, P., & Acikmese, B. (2021). Advances in trajectory optimization for space vehicle control.

Mao, Y., Szmuk, M., & Acikmese, B. (2016). Successive convexification of non-convex optimal control problems and its convergence properties. *2016 IEEE 55th Conference on Decision and Control (CDC)*. https://doi.org/10.1109/cdc.2016.7798816

Märtens, M., Farries, K., Culton, J., & Chin, T.-J. (2024). Synthetic lunar terrain: A multimodal open dataset for training and evaluating neuromorphic vision algorithms. https://arxiv.org/abs/2408.16971

Martin, I., Dunstan, M., & Gestido, M. S. (2019). Planetary surface image generation for testing future space missions with pangu. https://api.semanticscholar.org/CorpusID:209505511

McLeod, S., Meoni, G., Izzo, D., Mergy, A., Liu, D., Latif, Y., Reid, I., & Chin, T.-J. (2022). Globally optimal event-based divergence estimation for ventral landing.

Meditch, J. (1964). On the problem of optimal thrust programming for a lunar soft landing. *IEEE Transactions on Automatic Control*, *9*(4), 477–484. https://doi.org/10.1109/TAC.1964.1105758

Moreno-Martín, S., Ros, L., & Celaya, E. (2024). Collocation methods for second and higher order systems. *Autonomous Robots*, *48*(2). https://doi.org/10.1007/s10514-023-10155-z

Mueggler, E., Rebecq, H., Gallego, G., Delbruck, T., & Scaramuzza, D. (2017). The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *The International Journal of Robotics Research*, *36*(2), 142–149. https://doi.org/10.1177/0278364917691115

Nagata, J., & Aoki, Y. (2022). Self-supervised learning of inlier events for event-based optical flow. https://bmvc2022.mpi-inf.mpg.de/0785.pdf

Neftci, E. O., Mostafa, H., & Zenke, F. (2019). Surrogate gradient learning in spiking neural networks.

Nielsen, M. A. (2018). *Neural networks and deep learning*. Determination Press. http://neuralnetworksanddeeplearning.com/

Orchard, G., Benosman, R., Etienne-Cummings, R., & Thakor, N. (2013). A spiking neural network architecture for visual motion estimation, 298–301. https://doi.org/10.1109/BioCAS.2013.6679698

Orchard, G., Jayawant, A., Cohen, G., & Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. https://arxiv.org/abs/1507.07629

Paredes-Vallés, F., Hagenaars, J., Dupeyroux, J., Stroobants, S., Xu, Y., & de Croon, G. (2023). Fully neuromorphic vision and control for autonomous drone flight.

Paredes-Vallés, F., Scheper, K. Y. W., & de Croon, G. C. H. E. (2018). Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception. *CoRR*, *abs/1807.10936*. http://arxiv.org/abs/1807.10936

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. https://arxiv.org/abs/1912.01703

Pehle, C., & Pedersen, J. E. (2021). Norse - A deep learning library for spiking neural networks. https://doi.org/10.5281/zenodo.4422025

Phung, V., & Rhee, E. (2018). A deep learning approach for classification of cloud image patches on small datasets. *Journal of Information and Communication Convergence Engineering*, *16*, 173–178. https://doi.org/10.6109/jicce.2018.16.3.173

Podareanu, D., Codreanu, V., Aigner, S., Leeuwen, C., & Weinberg, V. (2019). Best practice guide - deep learning. https://doi.org/10.13140/RG.2.2.31564.05769

Ponghiran, W., Liyanagedera, C. M., & Roy, K. (2023). Event-based temporally dense optical flow estimation with sequential learning. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 9827–9836.

Posch, C., Matolin, D., & Wohlgenannt, R. (2011). 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. https://doi.org/10.1109/JSSC.2010.2085952

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*, 1470–1484. https://doi.org/10.1109/JPROC.2014.2346153

Potmesil, M., & Chakravarty, I. (1983). Modeling motion blur in computer-generated images. *ACM SIGGRAPH Computer Graphics*, *17*, 389–399. https://doi.org/10.1145/964967.801169

Rançon, U., Cuadrado-Anibarro, J., Cottereau, B. R., & Masquelier, T. (2021). Stereospike: Depth learning with a spiking neural network. *CoRR*, *abs/2109.13751*. https://arxiv.org/abs/2109.13751

Ranjan, C. (2020). Theory of pooling. https://doi.org/10.13140/RG.2.2.23408.07688

Ravasio, C. S., Da Cruz, L., & Bergeles, C. (2021). Oflibnumpy & oflibpytorch: Optical flow handling and manipulation in python. *Journal of Open Research Software (JORS)*, *9*. https://doi.org/10.5334/jors.380

Rebecq, H., Gehrig, D., & Scaramuzza, D. (2018). Esim: An open event camera simulator (A. Billard, A. Dragan, J. Peters, & J. Morimoto, Eds.). *87*, 969–982. https://proceedings.mlr.press/v87/rebecq18a.html

Reynolds, T. P., & Mesbahi, M. (2020). Optimal planar powered descent with independent thrust and torque. *Journal of Guidance Control and Dynamics*, *43*, 1225–1231. https://api.semanticscholar.org/CorpusID:219741752

Roffe, S., Akolkar, H., George, A. D., Linares-Barranco, B., & Benosman, R. B. (2021). Neutron-induced, single-event effects on neuromorphic event-based vision sensor: A first step and tools to space applications. *IEEE Access*, *9*, 85748–85763. https://doi.org/10.1109/access.2021.3085136

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation.

Roy, K., Jaiswal, A. R., & Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature*, *575*, 607–617. https://api.semanticscholar.org/CorpusID:208329736

Scheerlinck, C., Rebecq, H., Gehrig, D., Barnes, N., Mahony, R. E., & Scaramuzza, D. (2020). Fast image reconstruction with an event camera, 156–163. https://doi.org/10.1109/WACV45572.2020.9093366

Schnider, Y., Wozniak, S., Gehrig, M., Lecomte, J., von Arnim, A., Benini, L., Scaramuzza, D., & Pantazi, A. (2023). Neuromorphic optical flow and real-time implementation with event cameras.

Schuman, C., Kulkarni, S., Parsa, M., Mitchell, J., Date, P., & Kay, B. (2022). Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, *2*, 10–19. https://doi.org/10.1038/s43588-021-00184-y

Serrano-Gotarredona, T., Park, J., Linares-Barranco, A., Jiménez, A., Benosman, R., & Linares-Barranco, B. (2013). Improved contrast sensitivity dvs and its application to event-driven stereo vision, 2420–2423. https://doi.org/10.1109/ISCAS.2013.6572367

Shatz, C. (1992). The developing brain. *Scientific American*, *267*, 60–7. https://doi.org/10.1038/scientificamerican0992-60

Singh, G. S., & Acerbi, L. (2024). PyBADS: Fast and robust black-box optimization in Python. *Journal of Open Source Software*, *9*(94), 5694. https://doi.org/10.21105/joss.05694

Stengel, R. F. (1969). Manual attitude control of the lunar module. *American Institute of Aeronautics and Astronautics Conference Proceedings*.

Stopar, J., & Meyer, H. (2019). Topographic map of the moon's south pole (85°s to pole).

Sun, H., Dao, M., & Frémont, V. (2022). 3d-flownet: Event-based optical flow estimation with 3d representation. *CoRR*, *abs/2201.12265*. https://arxiv.org/abs/2201.12265

Sung, M., & Kim, Y. (2020). Training spiking neural networks with an adaptive leaky integrate-and-fire neuron, 1–2. https://doi.org/10.1109/ICCE-Asia49877.2020.9277455

Thurman, S. (2004). Surveyor spacecraft automatic landing system.

Tian, Y., & Andrade-Cetto, J. (2023). Egomotion from event-based snn optical flow. https://doi.org/10.1145/3589737.3605978

Tian, Y., & Andrade-Cetto, J. (2024). Sdformerflow: Spatiotemporal swin spikeformer for event-based optical flow estimation. https://arxiv.org/abs/2409.04082

Tomar, S. (2006). Converting video formats with ffmpeg. *Linux Journal*, *2006*(146), 10.

Valette, F., Ruffier, F., Viollet, S., & Seidl, T. (2010). Biomimetic optic flow sensing applied to a lunar landing scenario. *Proceedings - IEEE International Conference on Robotics and Automation*, 2253–2260. https://doi.org/10.1109/ROBOT.2010.5509364

Venkatesan, R., & Li, B. (2018). *Convolutional neural networks in visual computing: A concise guide*. CRC Press. https://books.google.nl/books?id=Y2xSAQAACAAJ

Wächter, A., & Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*, 25–57. https://doi.org/10.1007/s10107-004-0559-y

Wang, Z., & Grant, M. (2016). Constrained trajectory optimization for planetary entry via sequential convex programming. https://doi.org/10.2514/6.2016-3241

Warren, W., Kay, B., Zosh, W., Duchon, A., & Sahuc, S. (2001). Optic flow is used to control human walking. *Nature neuroscience*, *4*, 213–6. https://doi.org/10.1038/84054

Wei, G., Li, X., Zhang, W., Tian, Y., Jiang, S., Wang, C., & Ma, J. (2023). Illumination conditions near the moon's south pole: Implication for a concept design of china's chang'e7 lunar polar exploration. *Acta Astronautica*, *208*, 74–81. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85151536620&doi=10.1016%2fj.actaastro.2023.03.022&partnerID=40&md5=7eedfd38b0a1f50625d20e7cab77a721

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proc. IEEE*, *78*, 1550–1560. https://api.semanticscholar.org/CorpusID:18470994

Yamazaki, K., Vo, K., Bulsara, D., & Le, N. (2022). Spiking neural networks and their applications: A review. *Brain sciences*, *12*. https://doi.org/10.3390/brainsci12070863

Yang, F., Su, L., Zhao, J., Chen, X., Wang, X., Jiang, N., & Hu, Q. (2023). Sa-flownet: Event-based self-attention optical flow estimation with spiking-analogue neural networks. *IET Computer Vision*, *17*, n/a–n/a. https://doi.org/10.1049/cvi2.12206

Yingge, H., Ali, I., & Lee, K.-Y. (2020). Deep neural networks on chip - a survey, 589–592. https://doi.org/10.1109/BigComp48618.2020.00016

Zhang, S., Wimmer-Schweingruber, R., Yu, J., Wang, C., Fu, Q., Zou, Y., Sun, Y., Wang, C., Hou, D., Böttcher, S., Burmeister, S., Seimetz, L., Schuster, B., Knierim, V., Shen, G., Yuan, B., Lohf, H., Guo, J., Xu, Z., & Quan, Z. (2020). First measurements of the radiation dose on the lunar surface. *Science advances*, *6*. https://doi.org/10.1126/sciadv.aaz1334

Zhang, Y., Lv, H., Zhao, Y., Feng, Y., Liu, H., & Bi, G. (2023). Event-based optical flow estimation with spatio-temporal backpropagation trained spiking neural network. *Micromachines*, *14*(1). https://doi.org/10.3390/mi14010203

Zhang, Y., Zhao, Y., Lv, H., Feng, Y., Liu, H., & Han, C. (2022). Adaptive slicing method of the spatiotemporal event stream obtained from a dynamic vision sensor. *Sensors*, *22*, 2614. https://doi.org/10.3390/s22072614

Zhang, Z., Cui, S., Chai, K., Yu, H., Dasgupta, S., Mahbub, U., & Rahman, T. (2023). V2ce: Video to continuous events simulator.

Zhou, Z., Siddiquee, M. M. R., Tajbakhsh, N., & Liang, J. (2020). Unet++: Redesigning skip connections to exploit multiscale features in image segmentation.

Zhu, A. Z., Thakur, D., Ozaslan, T., Pfrommer, B., Kumar, V., & Daniilidis, K. (2018). The multivehicle stereo event camera dataset: An event camera dataset for 3d perception. *IEEE Robotics and Automation Letters*, *3*(3), 2032–2039. https://doi.org/10.1109/lra.2018.2800793

Zhu, A. Z., Wang, Z., Khant, K., & Daniilidis, K. (2019). Eventgan: Leveraging large scale image datasets for event cameras. *CoRR*, *abs/1912.01584*. http://arxiv.org/abs/1912.01584

Zhu, A. Z., Yuan, L., Chaney, K., & Daniilidis, K. (2018a). Ev-flownet: Self-supervised optical flow estimation for event-based cameras. *ArXiv*, *abs/1802.06898*. https://api.semanticscholar.org/CorpusID:3396150

Zhu, A. Z., Yuan, L., Chaney, K., & Daniilidis, K. (2018b). Unsupervised event-based learning of optical flow, depth, and egomotion. *CoRR*, *abs/1812.08156*. http://arxiv.org/abs/1812.08156