ABSTRACT

Title of dissertation:	MOTION SEGMENTATION AND EGOMOTION ESTIMATION WITH EVENT-BASED CAMERAS
	Anton Mitrokhin Doctor of Philosophy, 2020
Dissertation directed by:	Prof. Yiannis Aloimonos and Dr. Cornelia Fermüller Department of Computer Science

Computer vision has been dominated by classical, CMOS frame-based imaging sensors for many years. Yet, motion is not well represented in classical cameras and vision techniques - a consequence of traditional vision being frame-based and only existing 'in the moment' while motion is a continuous entity. With the introduction of neuromorphic hardware, such as the event-based cameras, we are ready to cross the bridge of frame based vision and develop a new concept - motion-based vision. The event-based sensor provides dense temporal information about changes on the scene - it can 'see' the motion at an equivalent of almost infinite framerate, making a perfect fit for creating dense, long term motion trajectories and allowing for a significantly more efficient, generic and at the same time accurate motion perception.

By its design, an event-based sensor accommodates a large dynamic range, provides high temporal resolution and low latency – ideal properties for applications where high quality motion estimation and tolerance towards challenging lighting conditions are desirable. The price for these properties is indeed heavy - event-based sensors produce a lot of noise, their resolution is relatively low and their data - typically referred to as *event cloud* - is asynchronous and sparse. Event sensors offer new opportunities for robust visual perception so much needed in autonomous robotics, but challenges associated with the sensor output, such as high noise, relatively low spatial resolution and sparsity, ask for different visual processing approaches.

In this dissertation we develop methods and frameworks for motion segmentation and egomotion estimation on event-based data, starting with a simple optimization-based approach for camera motion compensation and object tracking and continuing by developing several deep learning pipelines, while continuing to explore the connection between the shapes of the event clouds and scene motion. We collect EV-IMO - the first pixelwise-annotated dataset for motion segmentation for event cameras and propose a 3D graph-based learning approach for motion segmentation in (x, y, t) domain. Finally we develop a set of mathematical constraints for event streams which leverage their temporal density and connect the shape of the event cloud with camera and object motion.

MOTION SEGMENTATION AND EGOMOTION ESTIMATION WITH EVENT-BASED CAMERAS

by

Anton Mitrokhin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Table of Contents

List of Tables v

List of Figures vi

1 Introduction 1
- 1.1 The Dynamic Vision Sensor . . . 2
- 1.2 Egomotion and Motion Segmentation . . . 5
- 1.3 Challenges of Motion Segmentation . . . 7
- 1.4 The Organization of the Thesis . . . 9

2 Related Work 10

3 Event-based Moving Object Detection and Tracking 13
- 3.1 Motivation . . . 13
- 3.2 Methods . . . 14
- 3.3 Preliminaries . . . 15
  - 3.3.1 Notation . . . 15
  - 3.3.2 Event Count Image . . . 16
  - 3.3.3 Time-image $\mathcal{T}$ . . . 17
  - 3.3.4 Minimization Constraints . . . 19
- 3.4 Camera Motion Compensation . . . 19
  - 3.4.1 Coarse Global Motion Minimization on $\mathcal{T}$ . . . 21
  - 3.4.2 Fine Global Motion Refinement on $\mathcal{I}$ . . . 21
- 3.5 Multiple Object Detection And Tracking . . . 22
  - 3.5.1 Detection . . . 23
  - 3.5.2 Tracking . . . 23
- 3.6 Datasets and Evaluation . . . 24
  - 3.6.1 Dataset collection . . . 25
  - 3.6.2 Computation Times . . . 25
  - 3.6.3 Metrics and Evaluation . . . 27
- 3.7 Conclusions . . . 28

4 Learning Sensori-Motor Control with Neuromorphic Sensors Using Hyperdimensional Binary Vectors 37
4.1 Methods 37
4.2 Implementation 38
4.3 Results 38

5 EVIMO: A Dataset for Motion Segmentation with Event-Based Cameras 42
5.1 Introduction 42
5.2 EV-IMO Dataset 43
5.2.0.1 Methodology 44
5.2.0.2 Dataset Generation 45
5.3 Sequences 47
5.4 Results on Learning Pipelines 47
5.4.1 Qualitative Results 48
5.4.2 Motion Estimation and Segmentation 49
5.4.3 Depth Estimation 50

6 Learning motion segmentation in (x, y, t) space using edge convolutional neural networks 52
6.1 Introduction 52
6.2 Event Clouds and Scene Motion 55
6.2.1 Surface Normals, Normal Flow, & Optical Flow 55
6.2.2 Continuity in Time 58
6.2.3 Boundary Regions and Occlusions 58
6.3 The Architecture 59
6.3.1 Motion Segmentation in 3D 59
6.3.2 Network Design 60
6.3.3 Edge Computation 61
6.3.4 Temporal Augmentation 62
6.4 Experiments 63
6.4.1 Dataset 63
6.4.1.1 Data Preprocessing 65
6.4.2 Implementation Details 65
6.4.3 Ablation Studies 66
6.4.3.1 Effects of the Slice Width 66
6.4.3.2 Temporal Augmentation 67
6.4.4 Results 68
6.4.4.1 Qualitative Results 69
6.4.4.2 Performance Considerations 70
6.5 Conclusion 70

7 Event-based constraints on camera egomotion, event surface normals and depth 72
7.1 Introduction 72
7.1.1 Motion Estimation and Event Cloud Structure 72
7.2 Preliminaries 73

7.2.1 Event Surface Parameterization 73
7.2.2 Notation 74
7.3 Constraint on Event Surface Normals 75
7.4 Optical Flow Equation 75
7.5 Constraint on the Second Order Derivatives 76
7.5.1 Derivative of Optical Flow 77
7.5.2 Derivative of Surface Normal Constraint 78
7.5.3 Second Order Constraint on Event Surfaces 78
7.6 Experiments 78
7.6.1 Dataset 78
7.6.2 Error Functions 79
7.6.2.1 Depth Positivity Error 80
7.6.2.2 Photometric Error 80
7.6.2.3 Variance Error 81
7.6.2.4 Second Order Error 82
7.6.3 Results 83
7.7 Camera Motion Model 84
7.7.1 Derivations for Angular Velocity and Acceleration 85

A Learning Visual Motion Segmentation using Event Surfaces - Additional Experiments 90
A.1 Edge Configurations - Ablation Study 91
A.2 Training Time 92
A.3 Additional Qualitative Results 92
A.3.1 Dataset 92
A.3.2 Inference with Large w 93
A.3.3 Additional Qualitative Results 93

B Towards Hyperdimensional Active Perception 98
B.1 Introduction 98
B.2 The Properties of Hyperdimensional Binary Vectors (HBVs) 103
B.3 Numerical vs. Categorical Values 108
B.4 Representing Numerical and Categorical Data 108
B.5 Encoding images as HBVs 112
B.6 Creating memories with HBVs 115
B.7 Experiments 116
B.7.1 Using neuromorphic visual information 116
B.7.2 Learning with HBVs 117
B.7.3 Perception to action binding with HBVs 118
B.7.4 Information capacity of HBVs 120
B.7.4.1 Theoretical limits on capacity of HBVs 121
B.7.5 Sensorimotor representation through binding visual and motor information 123

Bibliography 124

List of Tables

1.1 A comparison of specifications for several event-based cameras 4

3.1 Evaluation of the motion compensation pipeline on the EED 27

4.1 Quantitative results for the MVSEC dataset with HBV inference 40

5.1 EV-IMO sequences 49
5.2 Evaluation on segmentation and motion estimation 51
5.3 Evaluation of the depth estimation 51

6.1 Motion segmentation results on EVIMO dataset with 3D network 67
6.2 Forward time for different uniform downsampling and temporal slice widths of event cloud 71

A.1 Average time per training step 92

List of Figures

1.1 Event-based pixel 2
1.2 Scene motion via events 5
1.3 The event-based output 6
1.4 An example of a complex scene for segmentation 8

3.1 Parametric model-fitting by alignment 30
3.2 Motion compensation pipeline 31
3.3 An example output of the motion compensating algorithm 32
3.4 Object detection on compensated frames 33
3.5 Detection tolerance to noise 33
3.6 Detection tolerance to high texture 34
3.7 Drone used in the dataset collection 34
3.8 Failure cases 35
3.9 Detection results on the EED dataset 36

4.1 HBV inference results 41

5.1 An example from EVIMO dataset 43
5.2 The main interface of the automatic annotation tool 45
5.3 Types of background geometry featured in the EV-IMO dataset 48
5.4 Qualitative results from our evaluation 50

6.1 Event cloud shape properties 53
6.2 Local 3D properties of event clouds 56
6.3 Architecture of the 3D segmentation Network 57
6.4 Edge configurations for segmentation network 62
6.5 Qualitative results on 3D motion segmentation 64
6.6 Comparison of inference performance vs. time slice widths 68
6.7 $mIoU$ scores on the validation set without temporal augmentation 69

7.1 Self-supervised inference with event-based constraints 83
7.2 Dataset for egomotion estimation collected at 640×480 resolution 87
7.3 Preprocessing of event-based data for event-based constraints 88

7.4 Error function behavior vs. linear velocity estimate 89

A.1 The effects of using edges computed within different radii r 90
A.2 EVIMO dataset in (x, y, t) space 94
A.3 Ablation study for different radii of the sphere around the event 95
A.4 Results for 3D inference to validate the invariance to slice width 96
A.5 Additional qualitative results for 3D inference 97

B.1 Tension minimization with HBVs 111
B.2 A visualization of intensity minimization 113
B.3 Hamming distance visualization of intensities 114
B.4 Spatial pixel encoding with HBVs 115
B.5 Composing image encodings with HBVs 115
B.6 DVS data visualization 117
B.7 Hamming distance decay 121

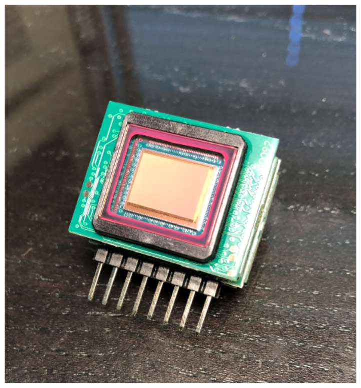
Chapter 1: Introduction

The recent advancements in imaging sensor development have outpaced the development of algorithms for processing image data. Recently, the computer vision community has started to derive inspiration from the neuromorphic community whose ideas are based on biological systems to build robust and fast algorithms which run on limited computing power.

The most challenging problems are encountered during scenarios requiring the processing of very fast motion with real-time control of a system. One such scenario is encountered in autonomous navigation. Although computer vision and robotics communities have put forward a solid mathematical frameworks and have developed many practical solutions, these solutions are currently not sufficient to deal with scenes with very high speed motion, high dynamic range, and changing lighting conditions. These are the scenarios where the event based frameworks excel.

Event-based vision sensors, such as the Dynamic Vision Sensor (DVS), are ideally suited for real-time motion analysis. The unique properties encompassed in the readings of such sensors provide high temporal resolution, superior sensitivity to light and low latency. These properties provide the grounds to estimate motion extremely reliably in the most sophisticated scenarios but they come at a price - modern event-based vision sensors have a lower resolution, compared to the classical cameras, sparse output and produce a significant amount of noise. Moreover, the asynchronous nature of the event stream is not compatible with traditional frame-based processing techniques, which calls for development of novel algorithms and introduction of new paradigms in event-based computer vision.

1.1 The Dynamic Vision Sensor



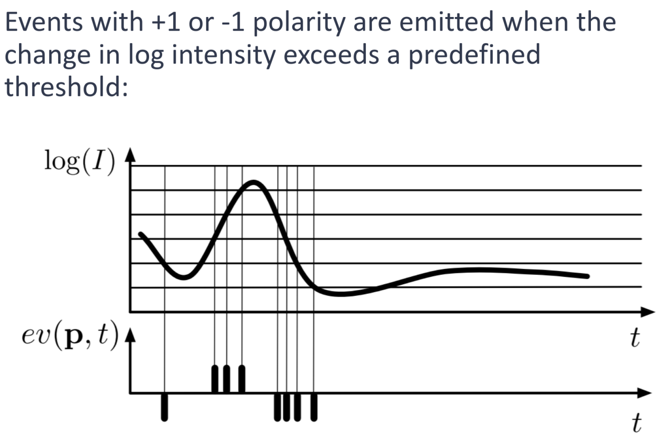

Events with +1 or -1 polarity are emitted when the change in log intensity exceeds a predefined threshold:
$\log(I)$
$ev(\mathbf{p}, t)$
t
t

Figure 1.1: *Event-based cameras are CMOS sensors, similar to conventional frame-based sensors. Left - Prophesee 640x480 sensor; right - the idea behind event-based signal: for each pixel, if the light intensity changes by a certain preset logarithmic threshold, the timestamp is sent to the common bus. Event polarity specifies whether the light intensity has increased or decreased.*

Event-based cameras provide dense temporal information about changes on the scene - effectively acting like cameras with infinite frame rate; with every pixel acting as an independent electrical circuit such sensors are not driven by a common clock - every pixel reacts to motion independently: when the light intensity on a given pixel changes by a certain predefined percentage-threshold, that pixel would report the current timestamp t (at a nanosecond resolution), its coordinates (x, y) and whether the intensity of light has increased or decreased, which is referred to as *polarity* p - Fig. 1.1. A set of (x, y, t, p) is called an *event*, and is sent independently from each camera pixel over an asynchronous bus, resulting in a continuous stream of data called an *event stream*.

For an event camera, every visible moving edge in the scene produces a trail of events - an *event cloud*, which lies on a surface (called *event-surface* or *time-surface* [1]) in (x, y, t) space. The time-surface contains all the information about structure and motion, and the shape of the event clouds is constrained by the laws of physics and epipolar geometry. A certain shape of a cloud, even locally, might signify that two visible objects occlude each other, collide with each other, change their motion or move closer to the camera. An example of the event camera output is shown on Fig. 1.3 and Fig. 1.2 - where event trails from the camera motion and an independently moving object are visible.

With the growing enthusiasm towards technologies such as VR and gesture recognition, many companies [2–5] have started to invest in the development of event-based cameras. We provide example specifications for a number of modern sensors in Table 1.1. While easily distinguished by their high dynamic range and low latency, the resolution of event-based cameras has shown a steady increase over the years from 128×128 in 2008 to 1280×800 in 2019.

	Year	Resolution	Latency (μs)	Sensitivity (%)
iniVation DAVIS128	2008	128×128	12	17
iniVation DAVIS346	2017	346×260	20	14.3
Prophesee Gen3 CD	2017	640×480	40	12
Samsung Gen3	2018	640×480	50	15
Insightness Rino 3	2018	320×262	125	15
CelePixel Celex V	2019	1280×800	8	10

Table 1.1: *A comparison of specifications for several modern event-based cameras; Event-based cameras are a novel type of visual sensor and are being actively developed by the community and industry. For a full table please refer to [6].*

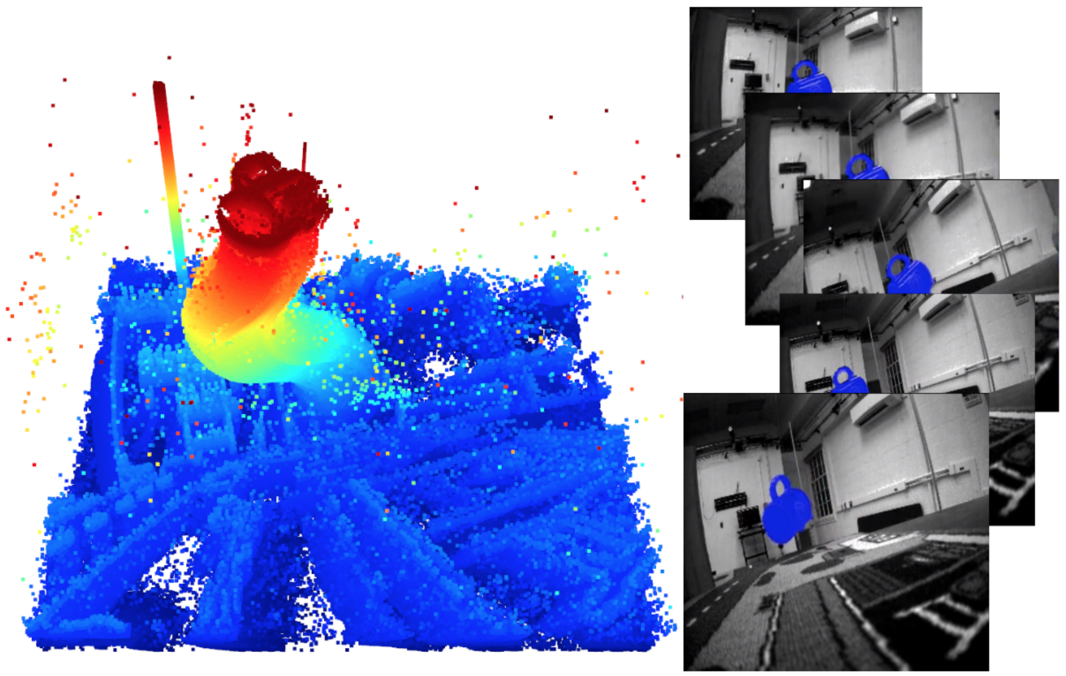


Figure 1.2: *Camera motion and and independently moving cup. Left - events; the color encodes the timestamps. Right, the classical frames with cup segmented in blue. The trajectory of the cup can be estimated by analyzing the shape of the event cloud. The camera moves only in the very beginning of the recording, leaving the cup clearly visible.*

1.2 Egomotion and Motion Segmentation

On the algorithmic side, the estimation of 3D motion and scene geometry has been of great interest in Computer Vision and Robotics for quite a long time, but most works considered a scene to be static. Earlier classical algorithms studied the problem of Structure from Motion (SfM) [7] to develop "scene independent" constraints (e.g. the epipolar constraint or depth positivity constraint) and estimate 3D motion from image to facilitate subsequent scene reconstruction. In recent years, most works have adopted the SLAM

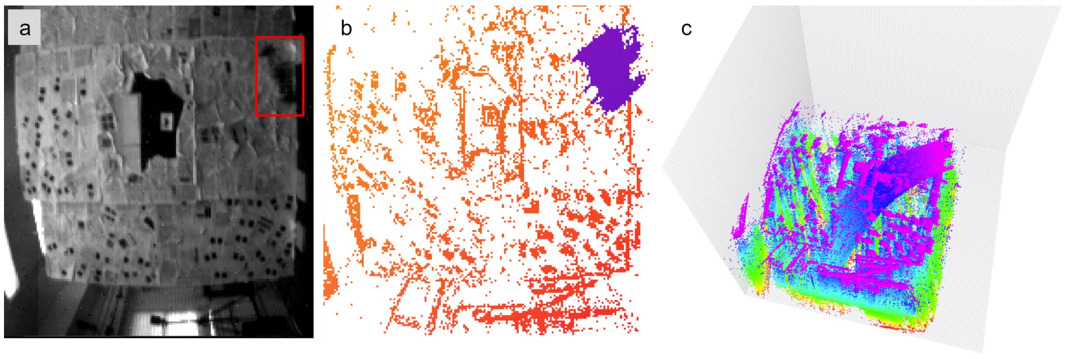


Figure 1.3: *Different motion representations acquired from the DAVIS [4] event-based sensor: (a) Grayscale image from an inbuilt frame-based camera (red bounding box denotes a moving object). (b) Motion-compensated projected event cloud. Color denotes inconsistency in motion. (c) The 3D representation of the event cloud in (x, y, t) coordinate space. Color represents the timestamp with [red - blue] corresponding to [0.0 - 0.5] seconds. The separately moving object (a quadrotor) is clearly visible as a trail of events passing through the entire 3D event cloud.*

philosophy [8], where depth, 3D motion and image measurements are estimated together using iterative probabilistic approaches, usually initialized with SfM estimates. Such reconstruction approaches are known to be computationally heavy and often fail in the presence of outliers.

Still, it is essential to emphasize the duality of the problems of motion segmentation, tracking and egomotion estimation. In essence, egomotion estimation can be thought of as an estimation of the motion and tracking of the ’largest cluster of points’ - although any rigid body on the scene could be chosen as a point of reference for the egomotion estimation.

In this light, we treat all our algorithms as ’Motion Estimation’ - and only use the term ’Egomotion Estimation’ in future chapters for simplicity. The motion segmentation provides the capability of choosing the desired point of reference by selecting a particular cluster, while from the point of view of the event camera the only difference between egomotion and motion of an independent object is the available spatial distribution of pixels, which typically allows to estimate egomotion more accurately than the motion of a small object.

1.3 Challenges of Motion Segmentation

It is important to stress a number of challenges which arise when attempting to perform visual segmentation with event-based data: event cameras are *differential sensors*, which means they are only capable of perceiving the changes in scene contrast, but not the structure of the scene per se. The meaningful information is only available on the edges and is sparse, and while conventional cameras are able to leverage the complimentary *appearance-induced* data modality, such as object color, event-based processing can only rely on contour motion. In our work we distinguish between recognition-based segmentation and motion segmentation and attempt to achieve the latter, since conceptually event cameras suit such tasks much better.

Another challenge is more fundamental and arises from the same fact - that the meaningful data is only available in regions with high contrast change. The contrast change by itself does not relate to motion directly - it can be induced by varying lighting and reflection; a shadow on the floor, for example, produces a false contour which might

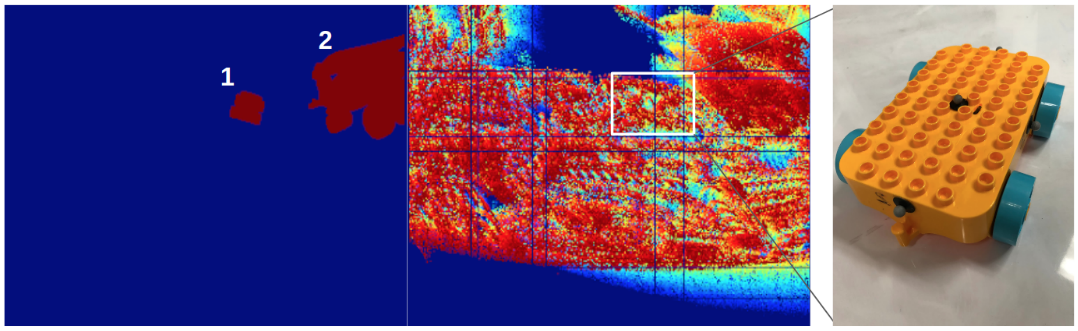


Figure 1.4: *An example of a complex scene for motion segmentation. The toy car (shown on the right) is moving across the carpet with high texture, while the camera is moving similarly to the object. On the left - the ground truth motion masks for object 1 (car) and 2 (plane). Middle - the 0.02 sec. slice of the event cloud; the car is almost not discernible from the background.*

change with the motion of the camera or independent object. In addition, low-contrast regions on the scene are sources of noise and create additional hurdles for event-based algorithms.

An example of a complex scene from EVIMO dataset (Chapter 5) is shown on Fig. 1.4. The depth variation between the background (high-textured carpet) and the foreground (toy car) is small in comparison to the distance to the camera; the velocity of the toy car in the world coordinate frame is small as well, making the magnitude of flow not induced by the egomotion small. This scene has few features to segment out the moving object and will require long-term event cloud analysis to solve reliably, which is discussed in Chapter 6.

1.4 The Organization of the Thesis

We develop our frameworks for event-based motion segmentation in several steps. First, in Chapter 3 we show how the accurate timestamp information of the events can be used for camera motion compensation and object detection without learning. We develop the 'Time Image' representation and use it in many subsequent works. We further expand on our findings in Chapter 4 by using a similar 4-parameter representation, this time to compute a 6dof egomotion in a self-driving car scenario using a novel learning pipeline.

To truly enable the leaning approaches on event-based data we develop the *first event-based motion segmentation dataset* with real data and pixelwise annotation. We discuss the dataset collection pipeline in Chapter 5; this is now a publicly released dataset, which allows the community to work on event-based segmentation for the first time without resorting to simulated scenarios.

We transition to deep learning in Chapter 6 and demonstrate a 3D motion segmentation method capable of taking advantage of event cloud geometry over large periods of time. We conclude by developing event camera constraints in Chapter 7; these constraints connect the camera motion and event surface geometry and allow to use the raw event cloud to directly optimize for scene depth or egomotion without the need for feature point detection or matching.

Chapter 2: Related Work

Scene motion analysis been studied for many years [9–11]. Lately, there has been an increased interest in these problems due to the applications in autonomous navigation [12] and robotics. The basic image representation for motion analysis is optical flow, representing pixelwise motion between two moments in time, while for event-based sensors it can be extended to *instantaneous* optical flow.

While most works process event data by collapsing information into 2D image maps, a few approaches have adopted concepts from 3D processing. The best known flow techniques on event cameras compute normals to local time-surfaces [13–16] to estimate normal flow. A block matching algorithm by Liu [17] uses similar ideas but can produce full flow for corner regions. As discussed in [18], local event information is inherently ambiguous. To resolve the ambiguity Barranco et al. [18] proposed to collect events over a longer time intervals and compute the motion from the trace of events at contours. Recent motion compensation approaches use the temporal information as a third dimension within a slice of events to derive flow and local or global motion models [19, 20].

Event-based feature detection research pursues similar ideas, but more global in temporal domain. Early works used the continuity of the event stream to bridge the gap between classical camera frames [21]. Later, Zhu *et al.* . introduced a probabilistic, event-only approach to corner extraction [22]. Manderscheid *et al.* have significantly improved on existing methods by using deep learning [23]. Lagorce *et al.* [1, 24] addressed event cloud analysis in terms of time-surfaces by designing temporal features and demonstrated them in recognition tasks, and Chandrapla *et al.* [25] learned motion invariant space-time features. These works laid the foundations of the spatio-temporal feature analysis on event clouds and are precursors to the global event-cloud analysis.

Learning approaches on event data are quite numerous, with many authors emphasising the importance of encoding temporal information as input features for neural networks. To name a few, [26] learns optical flow by constructing a discretized map with event timestamps, similar to the average time image used in [27]. Most recently, Zhu *et al.* [26] released the MVSEC dataset [28] and proposed self-supervised learning algorithm to estimate optical flow. An improved work by Zhu [29] uses multiple slices as input, retaining 3D structure of the cloud better. Barranco *et al.* [30] used different local spatio-temporal features to learn borders of objects. In our previous work, we have released the EV-IMO pipeline and the dataset [31] for 6-DoF motion estimation and segmentation (Chapter 5).

In pioneering work, Saxena *et al.* [32] demonstrated that shape can be learned from single images, inspiring many other supervised depth learning approaches (e.g. [33]). The concept was recently adopted in the SfM pipeline, and used in stereo [34] and video [35]. Nevertheless, the majority of event-based depth estimation methods [36, 37] use two or more event cameras. The first works on event-based monocular depth estimation were presented in [38] and [39]. Rebecq *et al.* [38] used a space-sweep voting mechanism and maximization strategy to estimate semi-dense depth maps where the trajectory is known. Kim *et al*. [39] used probabilistic filters to jointly estimate the motion of the event camera, a 3D map of the scene, and the intensity image. More recently, Gallego *et al*. [40] proposed a unified framework for joint estimation of depth, motion and optical flow.

Many motion segmentation methods used in video applications are based on 2D measurements only [41, 42]. 3D approaches model the camera's rigid motion. Thompson and Pong [43] first suggested detecting moving objects by checking contradictions to the epipolar constraint. Vidal *et al*. [44] introduced the concept of subspace constraints for segmenting multiple objects. A good motion segmentation requires both constraints imposed by the camera motion and some form of scene constraints for clustering into regions. The latter can be achieved using approximate models of the rigid flow or the scene in view, for example by modeling the scene as planar, fitting multiple planes using the plane plus parallax constraint [45], or selecting models depending on the scene complexity [46]. In addition constraints on the occlusion regions [47] and discontinuities [48] have been used. Recently, machine learning techniques have been used for motion segmentation [49, 50].

Chapter 3: Event-based Moving Object Detection and Tracking

In this chapter we present an approach to object tracking with asynchronous cameras using a simple optimization framework. We present a novel event stream representation which enables us to utilize information about the dynamic (temporal) component of the event stream. The 3D geometry of the event stream is approximated with a parametric model to motion-compensate for the camera (without feature tracking or explicit optical flow computation), and then moving objects that don't conform to the model are detected in an iterative process - Fig. 3.1. We demonstrate our framework on the task of independent motion detection and tracking, where we use the temporal model inconsistencies to locate differently moving objects in challenging situations of very fast motion.

The dataset used in this work is available at: ***http://prg.cs.umd.edu/BetterFlow.html***

The C++ implementation can be found here: ***https://github.com/better-flow/better-flow***

We provide experimental Python bindings here: ***https://github.com/better-flow/pydvs***

3.1 Motivation

The most challenging problems are encountered during scenarios requiring the processing of very fast motion with real-time control of a system. One such scenario is encountered in autonomous navigation. Although computer vision and robotics communities have put forward a solid mathematical framework and have developed many practical solutions, these solutions are currently not sufficient to deal with scenes with very high speed motion, high dynamic range, and changing lighting conditions. These are the scenarios where the event based frameworks excel.

While the majority of works focus on adapting the existing deep learning pipelines to the new domain of event-based vision, we show that it is possible to directly utilize the information from event-based camera to achieve reasonable detection and tracking performance in a lightweight pipeline, without the shortcomings of learning approaches.

3.2 Methods

Our algorithm derives inspiration from 3D point cloud processing techniques, such as Kinect Fusion [51], which use warp fields to perform a global minimization on point clouds. The algorithm performs global motion compensation of the camera by fitting a 4-parameter motion model to the cloud of events in a small time interval. These four parameters are the the x-shift, y- shift, expansion, and 2D rotation denoted as (h_x, h_y, h_z, θ). We use two different error functions in the minimization in different stages of the algorithm, which are defined in equations 3.4 and 3.6. It can be shown that the 4-parameter model is a good approximation for rigid camera motion and fronto-parallel planar scene regions. The algorithm then looks for the event clusters which do not conform to the motion model and labels them as separately moving regions, while at the same time fitting the motion model to each of the detected regions.

Sec. 3.3.1 describes the general notation used in this chapter, and Secs. 3.3.2 and

3.3.3 provide an intuition for the error functions used for motion compensation. The details of the algorithm are described in Sec. 3.4 (motion compensation) and 3.5 (object detection and tracking).

3.3 Preliminaries

3.3.1 Notation

Let the input events in a temporal segment $[t_0, t_0 + \delta t]$ be represented by a 3-tuple $C\{x, y, t\} \in \mathbb{R}^3$. Here $\{x, y\}$ denote the spatial coordinates in the image plane and t denotes the event timestamps[1]. t_0 can be arbitrarily chosen, as the DVS data is continuous. However, the algorithm functions on a small time segment δt.

Let us denote the 2D displacement that maps events (x, y) at time t to their locations (x', y') at time t_0 by a warp field $\phi(x, y, t - t_0) : (x, y, t) \rightarrow (x', y', t)$. Our goal is to find the motion compensating warp field $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ such that the motion compensated events, when projected onto the image plane, have maximum density. Let us denote these motion compensated events as:

$$\begin{aligned} C' = \Pi\{\phi(C)\} &= \Pi\{\phi(x, y, t - t_0)\} \\ &= \{x', y', 0\} \quad \forall\{x, y, t\} \in C \end{aligned} \tag{3.1}$$

Due to the geometric properties of the events, such a warp field encodes the per-

[1]The data from the DVS sensor is four-dimensional, with the additional, fourth component a binary value denoting the sign of intensity change. However, because of noise at object boundaries, we do not utilize this value here.

event optical flow. Here Π is the temporal projection function projecting motion compensated events along the time axis. Π is used to reduce the dimensionality of the data from $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ and simplify the minimization process. We represent the data available in C with two discretized maps which encode the temporal and intensity properties of the event stream.

3.3.2 Event Count Image

To calculate the event density $\mathcal{D}$ of $C' = \Pi\{\phi(C)\}$, we discretize the image plane into pixels of a chosen size. We use symbols $(i, j) \in \mathbb{N}^2$ to denote the integer pixel (discretization bin) coordinates, while $(x', y', t) \in \mathbb{R}^3$ represent real-valued warped event coordinates. Each projected event from C' is mapped to a certain discrete pixel, and the **total number of events mapped to the pixel is recorded as a value of that pixel**. We will henceforth refer to this data structure as *event-count image* $\mathcal{I}$. Let

$$\xi_{ij} = \{\{x', y', t\} : \{x', y', 0\} \in C', \, i = x', \, j = y'\} \tag{3.2}$$

be the *proposed event trajectory* - a set of warped events along the temporal axis which get projected onto the pixel (i, j) after the ϕ operation has been applied. Then the event-count image pixel $\mathcal{I}_{ij}$ is defined as:

$$\mathcal{I}_{ij} = |\xi_{ij}| \tag{3.3}$$

Here, $|A|$ is the cardinality of the set A. The event density $\mathcal{D}$ is computed as:

$$\mathcal{D} = \frac{\sum_{i,j} \mathcal{I}_{ij}}{\#\mathcal{I}} = \frac{|C'|}{\#\mathcal{I}} \tag{3.4}$$

where, $\#\mathcal{I}$ denotes the number of pixels with at least one event mapped on it. Since $|C'|$ is a constant for a given time slice, the problem can be reformulated as the minimization of the *total area* $S = \#\mathcal{I}$ on the event-count image. A similar formulation was used in [52] to estimate the rotation of the camera with the acutance as an error metric, instead of the area.

3.3.3 Time-image $\mathcal{T}$

Keen readers would observe that the event-count image representation $\mathcal{I}$ suffers from a subtle drawback. When the projection operation is performed, events produced by different edges (edges corresponding to different parts of the same real-world object or different real-world objects) can get projected onto the same pixel. This is a very common situation which occurs during fast motion in highly textured scenes.

To alleviate this problem, we utilize the information from the event timestamps t by proposing a novel representation which we call the *time-image* $\mathcal{T}$. Similar to $\mathcal{I}$ described before, $\mathcal{T}$ is a discretized plane with each pixel containing the *average* timestamp of the events mapped to it by the warp field ϕ.

$$\mathcal{T}_{ij} = \frac{1}{\mathcal{I}_{ij}} \sum t : t \in \xi_{ij} \tag{3.5}$$

Computing the mean of timestamps allows us to increase fidelity of our results by making use of all available DVS events. An alternative approach would be to consider only the latest timestamps [26], where performance would suffer in low light situations. This is because the signal to noise ratio in DVS events depends on the average illumination, the smaller the average illumination, the larger the noise. Note that the value of a time-image pixel $\mathcal{T}_{ij}$ (or better its deviation from the mean value) correlates with the probability that the motion was not compensated locally - this will be used later for motion detection in Sec. 3.5.1. $\mathcal{T}$ follows the 3D structure of the event cloud, and its gradient (G) provides the global metrics of the motion-compensation error which will be minimized by the algorithm:

$$Error = \sum \|G[i,j]\| = \sum (G_x^2[i,j] + G_y^2[i,j]) \tag{3.6}$$

Here $(G_x[i,j], G_y[i,j])$ denote the local spatial gradient of $\mathcal{T}$ along the (x) or (y) axes. Equation 3.6 is a global error which takes into account local motion inconsistencies of the warped event cloud. Together with the rigid body motion assumption, equation 3.6 can be decomposed into equations:

$$d_x = \frac{\sum G_x[i,j]}{\#\mathcal{I}}, \qquad d_y = \frac{\sum G_y[i,j]}{\#\mathcal{I}} \tag{3.7}$$

$$d_z = \frac{\sum (G_x[i,j], G_y[i,j]) \cdot (i,j)}{\#\mathcal{I}} \tag{3.8}$$

$$d_\theta = \frac{\sum (G_x[i,j], G_y[i,j]) \times (i,j)}{\#\mathcal{I}} \tag{3.9}$$

Equations 3.7, 3.8 and 3.9 will be used to provide gradients for the motion compensation algorithm in Sec. 3.3.4 - each of them correspond to the error for one of the model parameters: $(h_x, h_y, h_z, h_\theta)$, which represent shift in the image plane, expansion and 2D rotation.

3.3.4 Minimization Constraints

The local gradients of $\mathcal{T}$ and the event density $\mathcal{D}$ both quantify the error in event cloud motion compensation. We model the global warp field $\phi^G(x, y, t)$ with a 4 parameter global motion model $\mathcal{M}^G = \{h_x, h_y, h_z, \theta\}$ to describe the distortion induced in the event cloud by the camera motion. The resulting coordinate transformation amounts to:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - t * \left[\begin{bmatrix} h_x \\ h_y \end{bmatrix} + (h_z + 1) * \begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix} \right] \tag{3.10}$$

Here the the original event coordinates $\{x, y, t\}$ are transformed into new coordinates $\{x', y', t\}$. Note that the timestamp remains unchanged in the transformation and is omitted in Eq. 3.10 for simplicity. Furthermore, we assume linear event trajectories ξ_{ij} (Eq. 3.2) within the time slice.

The parameters of the model denote the shift (h_x, h_y) parallel to the image plane, a motion (h_z) towards the image plane, effectively a radial expansion of the event cloud, and a rotation (θ) around the Z axis, effectively a circular component of the event cloud.

3.4 Camera Motion Compensation

Our pipeline (see Fig. 3.2) consists of camera motion compensation and subsequent motion inconsistency analysis to detect independently moving objects. To compensate for the global background motion, the four parameter model $\mathcal{M}^G$ presented in Sec. 3.3.4 is used. The background motion is estimated, and objects are detected. Then the background model is refined using data from the background region only (not the detected objects)

(Fig. 3.3), and four-parameter models are fit to the segmented objects for tracking.

As outlined in Secs. 3.3.2 and 3.3.4, $\mathcal{T}$ and $\mathcal{I}$ are local metrics of event cloud misalignment but based on different sources of data - *event timestamps* and *event rates*. $\mathcal{T}$ provides a poor error metric when the optimizer is close to the minima due to the event averaging scheme employed. In particular, the global error gradient functions (Eqs. 3.7 - 3.9) have very small values and are unreliable.

Note that, $\mathcal{T}$ provides reliable gradients of the parameters of model $\mathcal{M}^G$ even in the presence of noise and fast motion, when events from different edges overlap during projection (see Fig. 3.6).

For the aforementioned reasons, the global motion minimization is performed in two stages: coarse motion minimization on $\mathcal{T}$ and fine motion refinement on the $\mathcal{I}$.

Algorithm 1 Global motion compensation in event space using $\mathcal{T}$.

```
**Data:** $\mathcal{M}^G_{i-1}, C, d, \xi$
**Result:** $\mathcal{M}^G_i, C', \mathcal{T}$
$C' \leftarrow$ warpEventCloud$(C, \mathcal{M}^G_{i-1})$
$\mathcal{T} \leftarrow$ getTimestampImage$(C', d)$
$\mathcal{M}^G_i \leftarrow$ updateModel$(\mathcal{M}^G_{i-1}, \mathcal{T})$
**while** $||M^G_{i-1} - M^G_i||_2 > \xi$ **do**
    $C' \leftarrow$ warpEventCloud$(C, \mathcal{M}^G_i)$
    $\mathcal{T} \leftarrow$ getTimestampImage$(C', d)$
    $\mathcal{M}^G_{i-1} \leftarrow \mathcal{M}^G_i$
    $\mathcal{M}^G_i \leftarrow$ updateModel$(\mathcal{M}^G_i, \mathcal{T})$
**end**
```

3.4.1 Coarse Global Motion Minimization on $\mathcal{T}$

An algorithm for coarse motion compensation of the event cloud is presented in Algorithm 1.

The input is the previous model $\mathcal{M}_{i-1}^{G}$, original event cloud C, the discretization grid size d and accuracy parameter ξ. The *warpEventCloud* function applies the warp field ψ as per equation (3.10). The time-image $\mathcal{T}$ is then generated on the warped event cloud C' according to (3.5). Finally, *updateModel* computes gradient images G_x and G_y (a simple Sobel operator is applied) and the gradients for the motion model parameters $\mathcal{M}_{i}^{G}$ are computed with (3.7 - 3.9). The parameters of $\mathcal{M}_{i}^{G}$ are then updated in a gradient descent fashion. Here, the discretization parameter d has been chosen as 0.3 of the DVS pixel size.

3.4.2 Fine Global Motion Refinement on $\mathcal{I}$

An additional fine motion refinement is done by maximizing the density $\mathcal{D}$ (3.4) of the event-count image $\mathcal{I}$. The density $\mathcal{D}$ function does not explicitly provide the gradient values for a given model, so variations of model parameters are used to acquire the derivatives and perform minimization. The corresponding algorithm is given in Algorithm 2.

Algorithm 2 Global motion compensation in event space with event count image

```
Data: $\mathcal{M}^G_{i-1}, C, d, \xi$
Result: $\mathcal{M}^G_i, C', \mathcal{I}$
$\mathcal{M}^G_i \leftarrow \mathcal{M}^G_{i-1}$
$C' \leftarrow$ warpEventCloud$(C, \mathcal{M}^G_{i-1})$
$\mathcal{I} \leftarrow$ getEventCountImage$(C', d)$
$\mathcal{D} \leftarrow$ getEventDensity$(\mathcal{I})$
$\mathcal{D}' \leftarrow 0$
while $||\mathcal{D} - \mathcal{D}'|| > \xi$ do
    $\mathcal{D} \leftarrow \mathcal{D}'$
    for Parameter p in Model $\mathcal{M}^G_i$ do
        $\mathcal{D}' \leftarrow$ getEventDensityCloud$(C, d, \mathcal{M}^G_i + p)$
        if $\mathcal{D}' > \mathcal{D}$ then
            $\mathcal{M}^G_i \leftarrow \mathcal{M}^G_i + p$
            $C' \leftarrow$ warpEventCloud$(C, \mathcal{M}^G_i)$
            $\mathcal{I} \leftarrow$ getEventCountImage$(C', d)$
        end
    end
end
```

3.5 Multiple Object Detection And Tracking

In this section, we describe the approach of the detection of independently moving objects by observing the inconsistencies of $\mathcal{T}$. The detected objects are then tracked using a traditional Kalman Filter.

3.5.1 Detection

We use a simple detection scheme: We detect pixels as independently moving using a thresholding operation and then group pixels into objects using morphological operations. Each pixel $\{i, j\} \in \mathcal{T}$ is associated with a score $\rho(x_i, y_j) \in [-1, 1]$ defined in Eq. 3.11, which quantitatively denotes the misalignment of independently moving objects with respect to the background. ρ is used as a measure for classifying a pixel as either background $\mathcal{B}$ or independently moving objects $\mathcal{O}_k$.

$$\rho(i, j) = \frac{\mathcal{T}(i, j) - < \mathcal{T}_{i,j} >}{\triangle t} \tag{3.11}$$

with $<>$ denoting the mean. Now, let us define $\mathcal{B}$ and $\mathcal{O}_k$.

$$\mathcal{B} = \{(i, j) | \rho(i, j) \leq 0\} \qquad \mathcal{O} = \{(i, j) | \rho(i, j) > \lambda\} \tag{3.12}$$

here $\mathcal{O} = \mathcal{O}_1 \bigcup ... \mathcal{O}_n$ (n is the number of independently moving objects) and λ is a predefined minimum confidence values for objects to be classified as independently moving. To detect independently moving objects, we then group foreground pixels using simple morphological operations.

3.5.2 Tracking

The detection algorithm presented in Subsec. 3.5.1 runs in real-time (processing time $< \delta t$) and is quite robust. To account for missing and wrong detections, especially in the presence of occlusion, we employ a simple Kalman Filter with a constant acceleration model. For the sake of brevity, we only define the state ($\mathcal{X}_k$) and measurement vectors ($\mathcal{Z}_k$) for the k^{th} object.

$$\mathcal{X}_k = [\hat{x}_k, \hat{y}_k, h_x, h_y, h_z, \theta, \hat{u}_k, \hat{v}_k]^T \tag{3.13}$$

$$\mathcal{Z}_k = [\hat{x}_k, \hat{y}_k, h_x, h_y, h_z, \theta]^T \tag{3.14}$$

where $\{\hat{x}_k, \hat{y}_k\}$ represent the mean coordinates of $\mathcal{O}_k$, h_x, h_y, h_z, θ represent the model parameters of the object, which is obtained by motion-compensating $\mathcal{O}_k$ as described in 3.4 and $\hat{u}_k, \hat{v}_k$ represents the average velocity of the k^{th} object.

3.6 Datasets and Evaluation

The Extreme Event Dataset (EED) was collected using the DAVIS [4] sensor under two scenarios. First, it was mounted on a quadrotor (Fig. 3.7), and second in a hand-held setup to accommodate for a variety of non-rigid camera motions. The recordings feature objects of multiple sizes moving at different speeds in a variety of lighting conditions. We emphasize the ability of our pipeline to perform detection at very high rates, and include several sequences where the tracked object changes its speed abruptly.

We have collected over 30 recordings in total but the centerpiece of our dataset is the 'Strobe Light' sequence. In this sequence, a periodically flashing bright light in a dark room creates significant noise. At the same time an object, another quadrotor is moving in the room. This is a challenge for traditional visual systems, but our bio-inspired algorithm shows excellent performance by leveraging the high temporal event density of the DAVIS sensor.

3.6.1 Dataset collection

The dataset was collected using the DAVIS240B bio-inspired sensor equipped with a $3.3mm.$ lens with a horizontal and vertical field of view of $80°$. Most of the sequences were created in a hand-held setting. For the quadropter sequences, we modified a Qualcomm Flight$^{\text{TM}}$ [53] platform to connect the DAVIS240B sensor to the onboard computer and collect data in a realistic scenario.

The setup of the quadrotor+sensor platform can be seen in Fig. 3.7. The overall weight of the fully loaded platform is $\approx 500g.$ and it is equipped with the Snapdragon APQ8074 ARM CPU, featuring 4 cores with up to 2.3GHz frequencies.

3.6.2 Computation Times

On a single thread of Intel Core$^{\text{TM}}$ i7 3.2GHz processor, Algorithms 1 and 2 take $10ms$ and $7ms$ on average for a single iteration step. However, both algorithms are based on a warp-and-project $\Pi\{\phi(C)\}$ operation which is highly parallelizable and thus well fit for implementation on a Graphical Processing Unit (GPU) or a Field-Programmable Gate Array (FPGA) to acquire very low latency and high processing speeds.

While a low level hardware implementation is beyond the scope of this research, we tested a prototype of the algorithm on an NVIDIA Titan X Pascal$^{\text{TM}}$ GPU with CUDA acceleration. A single iteration for Algorithms 1 and 2 takes $0.01ms$ and $0.003ms$ on average, respectively, which is a 1000X and 2333X speed-up. We have empirically found that the minimization converges on average in less than 30 iterations which ensures a faster-than-real-time computation speed with a high margin.

The recordings are organized into several sequences according to the nature of scenarios present in the scenes. All recording feature a variety of camera motions, with both rotational and translational motion (See fig. 3.9):

- *"Fast Moving Drone"* - A sequence featuring a single small remotely controlled quadrotor. Quadrotor has a rich texture and moves across various backgrounds in daylight lighting conditions following a variety of trajectories.

- *"Multiple objects "* - This sequence consists of multiple recordings with 1 to 3 moving objects under normal lighting conditions. The objects are simple, some of them have little to no texture. The objects move at a variety of speeds, either along linear trajectories or they bump from a surface.

- *"Lighting variation"* - A strobe light flashing at frequencies of 1-2 Hz was placed in a dark room to produce a lot of noise in the event sensor. This is an extremely challenging sequence, otherwise similar to the *"Fast Moving Drone"*.

- *"What is a Background?"* - In most tracking algorithm evaluations, object moves in front of the background. The following toy sequence was included, to show that it is possible to track an object even when the background occupies the space in between the camera and the object: A simple object was placed behind the net and the motion could only be seen through the net. Recordings contain a variety of distances between the net and the camera and the object is thrown at different speeds.

Table 3.1: *Evaluation of the motion compensation pipeline on the EED*

Sequence	”Fast Moving Drone”	”Multiple objects ”	”Lighting variation”	”What is a Background?”	”Occluded Sequence”
Success Rate	**92.78%**	**87.32%**	**84.52%**	**89.21%**	**90.83%**

- *”Occluded Sequence”* - The sole purpose of this sequence is to test the reliability of tracking in scenarios when detection is not possible for a small period of time. Several recordings feature object motion in occluded scenes.

3.6.3 Metrics and Evaluation

We define our evaluation metrics in form of success rate: We have acquired the ground truth by hand labeling the RGB frames of the recordings. We then computed a separate success rate for every time slice corresponding to an RGB frame from the DAVIS sensor as the percentage of the detected objects with at least 50% overlap with the object visible in the RGB frame. The mean of those scores for all sequences is reported in Table 3.1.

Although we did not discover sequences where the motion-compensation pipeline performs poorly, the particular difficulty for the tracking algorithm were the strobe light scenes where noise from the strobe light completely covered the tracked object and prevented detection - the noise on such scenes was even further amplified by the low lighting conditions.

Interestingly, a high performance was achieved on the *”What is a Background?”* sequence. The object was partially occluded by the net (located between the camera and the object), but in favor for the algorithm, the high texture of the net allowed for a robust camera motion compensation.

other challenging time sequences were the ones which featured objects whose paths crossed in the image. The Kalman filter often was able to distinguish between the tracked objects based on the difference in states (effectively, the difference in the previous motion)

To conclude this section, we feel the need to discuss some common failure cases. Figure 3.8 demonstrates a frame from the ”Fast Moving Drone” sequence - the motion of the drone with respect to the background is close to zero. The motion-compensation stage successfully compensates the camera motion but fails to recognize a separately moving object at that specific moment of time.

Another cause for failure is the presence of severe noise, as demonstrated by the ”Lighting variation” sequence. The motion compensation pipeline is robust only if a sufficient amount of background is visible. Figure 3.8 (right image) demonstrates that in some conditions too much noise is projected on the time image which renders both motion compensation and detection stages unreliable.

3.7 Conclusions

We argue that event-based sensing can fill a void in the area of robotic visual navigation. Classical frame-based Computer and Robot vision has great challenges in scenes with fast motion, low-lighting, or changing lighting conditions. We believe that event-based sensing coupled with active purposive algorithmic approaches can provide the necessary solutions. Along this thinking, we have presented an event-based only method for motion segmentation under unconstrained conditions. The essence of the algorithm lies in a method for efficiently and robustly estimating the effects of 3D camera motion from the event stream. Experiments in challenging conditions of fast motion with multiple moving objects and lighting variations demonstrate the usefulness of the method.

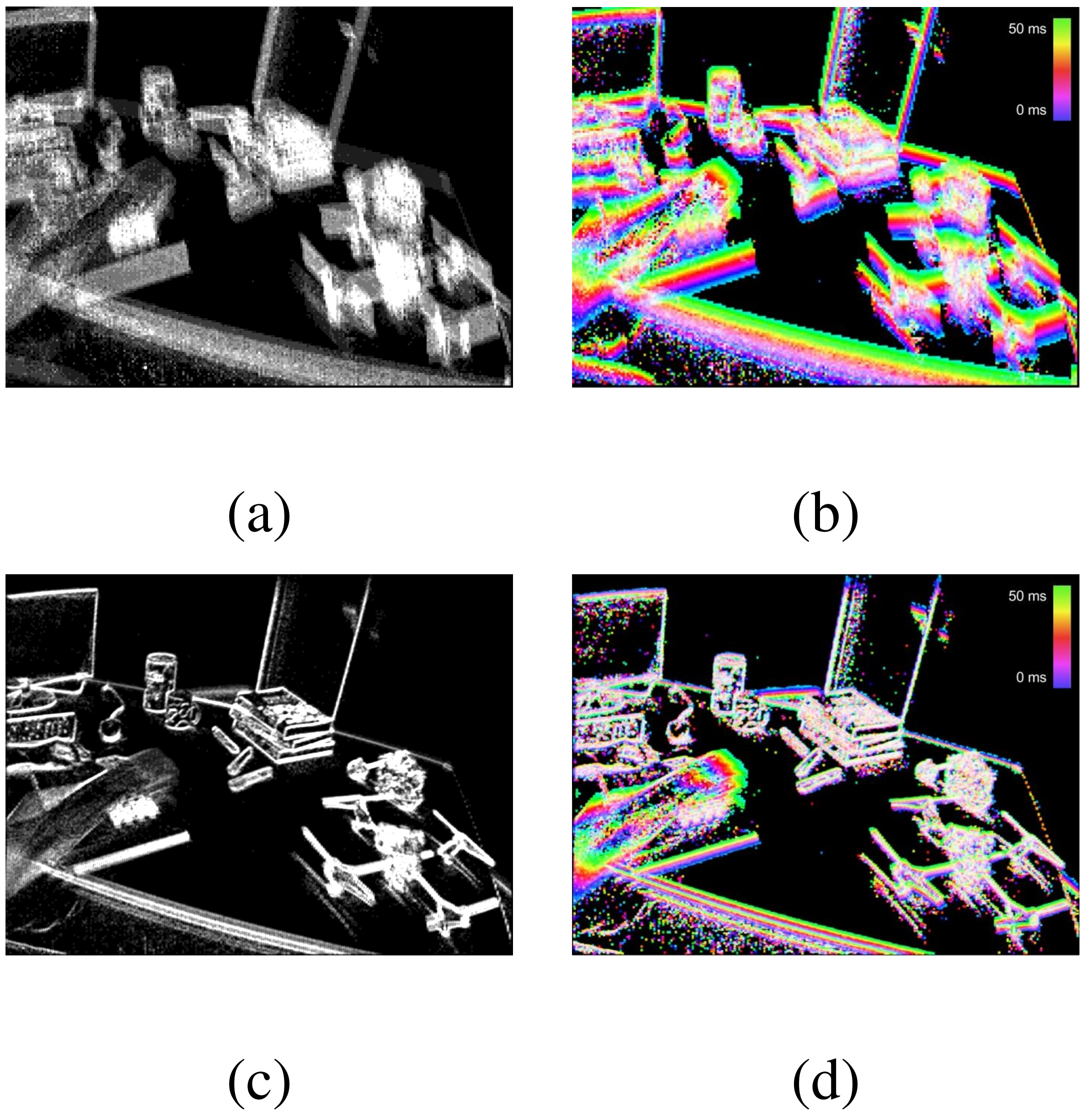


Figure 3.1: *Illustration of parametric model-fitting by alignment. The input is a cloud of events within a small time interval δt. Visualization of event counts (a), and average time stamps (b) mapped to the image pixels after reprojection. Motion-compensated event count (c) and time image (d) acquired after the minimization stage. Colors encode the event time stamp value (with blue for oldest and green for the most recent events). Note the hand that moves independently from the camera is clearly visible on the time image, which is the basis for the subsequent detection and tracking.*

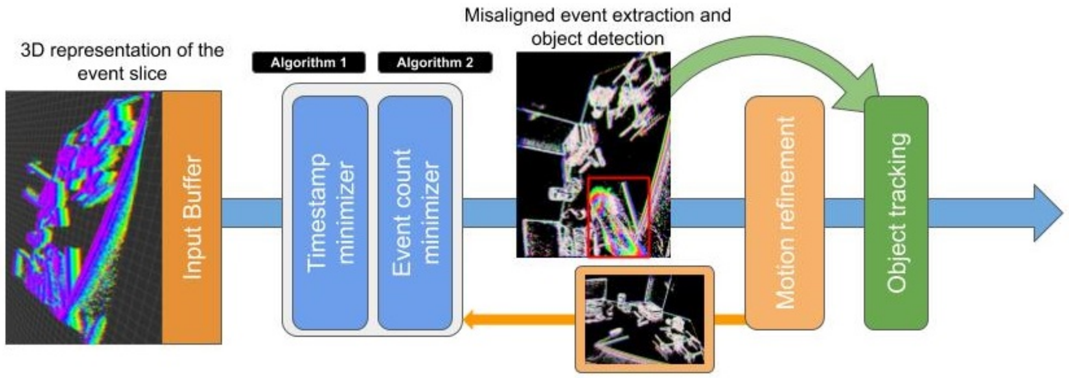


Figure 3.2: *Motion compensation and independent object tracking pipeline. The event cloud motion is minimized with the error functions defined on the time-image $\mathcal{T}$ and then, the motion is refined by maximizing the event density on the event-count image $\mathcal{I}$. The subsets of events with high probability of misalignment (shown in red bounding box) are removed and tracked and the remaining event cloud is again motion-compensated.*

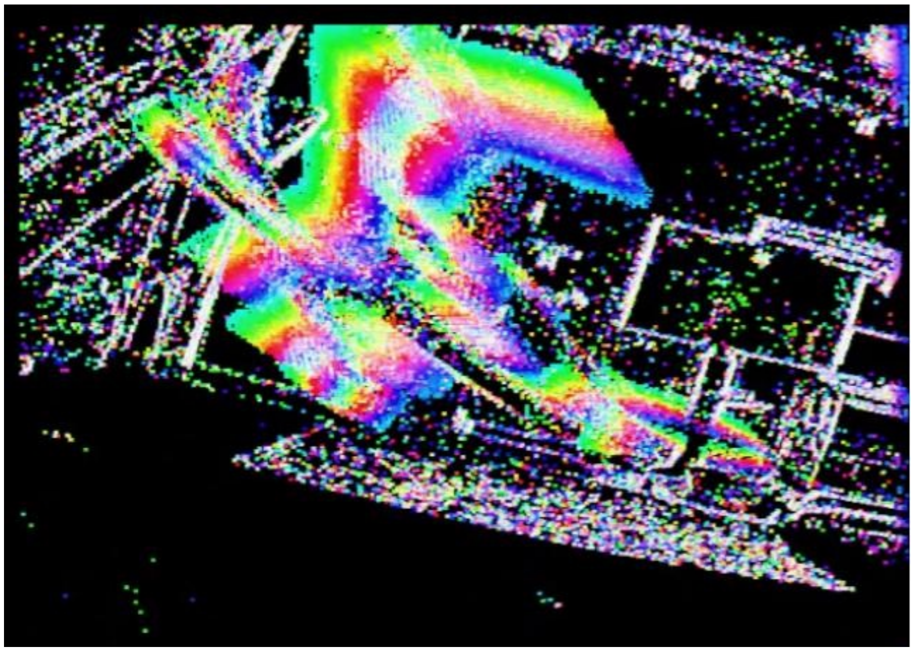


Figure 3.3: *An example output of the motion compensating algorithm - the time-image. The colors denote the average timestamps (blue is t_0, green is $t_0 + \delta t$). On this example the separately moving object (drone) occupies the large area of the frame but the camera motion compensation still succeeds.*

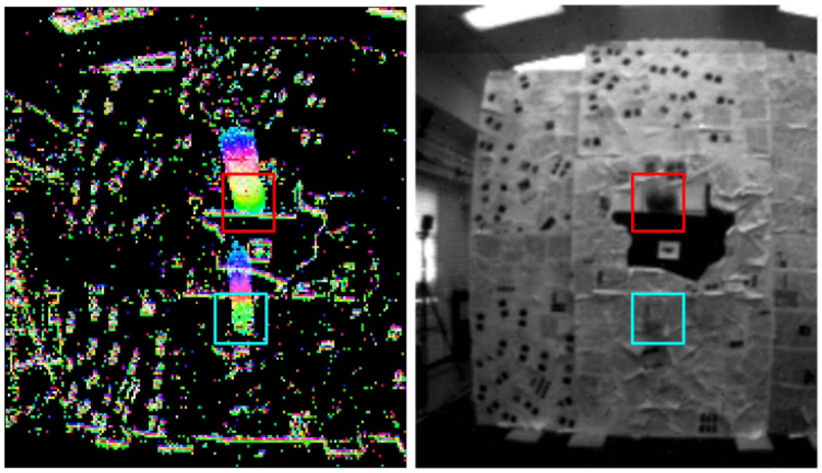
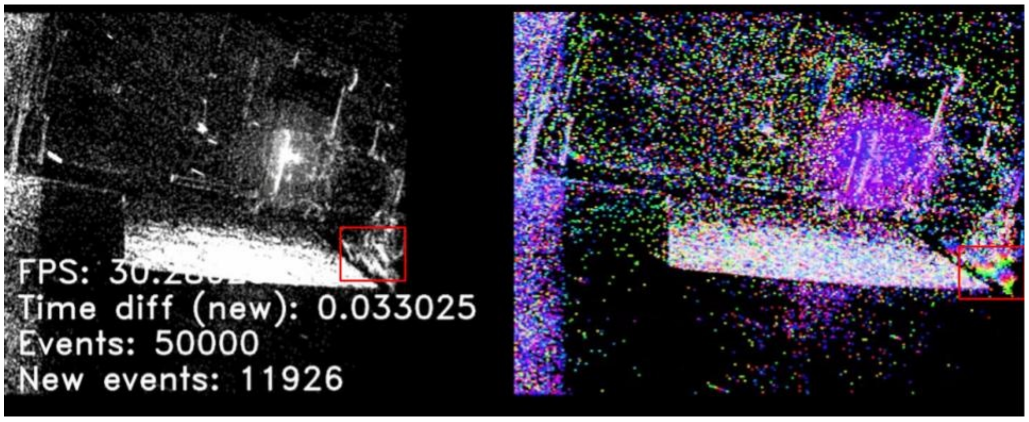


Figure 3.4: *A frame from the 'Two Objects' dataset. The left image shows the misalignment of the two objects after the global motion compensation (green color corresponds to the most recent events). The right image is the corresponding grayscale image - the objects (highlighted by bounding boxes) are poorly visible due to severe motion blur.*



```
FPS: 30.28
Time diff (new): 0.033025
Events: 50000
New events: 11926
```

Figure 3.5: *An example from the strobe dataset. A single object moves in a dark room with a bright strobe light which produces a lot of noise on the sensor output. Since the motion model is global, the minimization and detection are tolerant to such noise. The detection output is shown with a superimposed bounding box.*

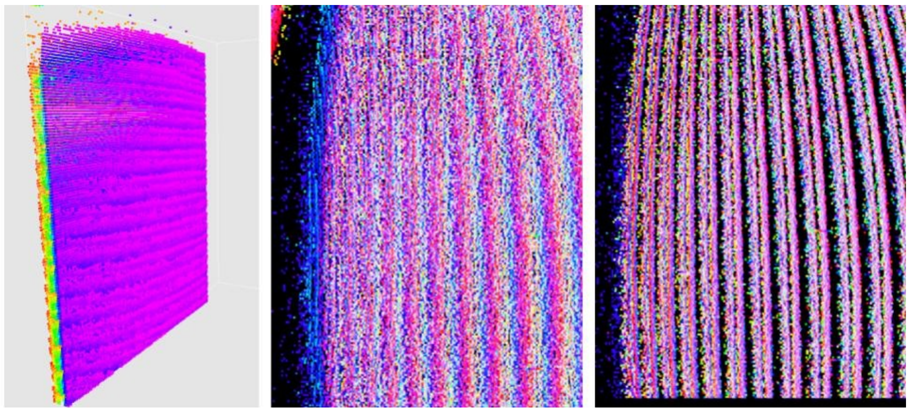


Figure 3.6: *An experiment with high texture and fast motion. The event cloud is a set of tilted planes which overlap during vertical projection. The time-image is a robust enough metric to compute the correct minimization gradient for the motion compensation. Images from left to right: 3D representation of event cloud, unminimized time-image, minimized time-image.*

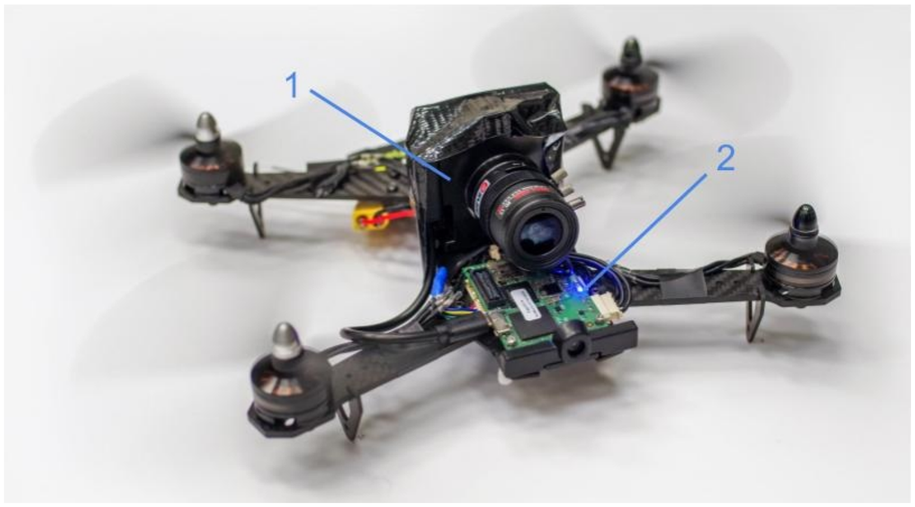


Figure 3.7: *Drone used in the dataset collection. 1 - mounted DAVIS240B camera, 2 - Customized Qualcomm Flight platform with onboard computer*

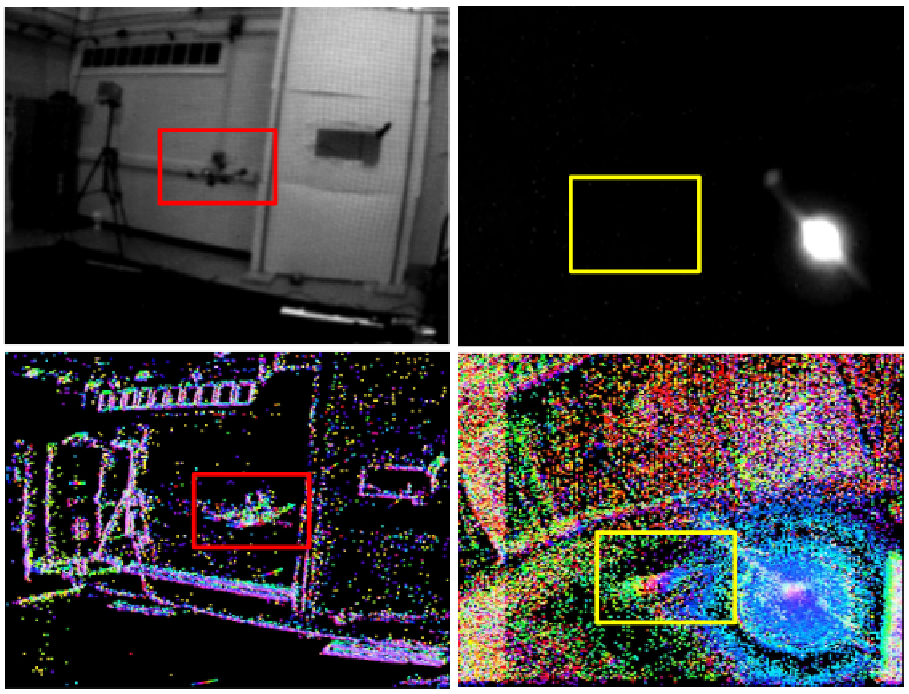


Figure 3.8: *Common failure cases. Top row: RGB frames, bottom row: time-images. Left: A setting featuring an object which does not move with respect to the background - a failure case for the detection stage, even though the edges are visible. Right: High noise scenario with the tracked object not visible in the gray-scale image.*

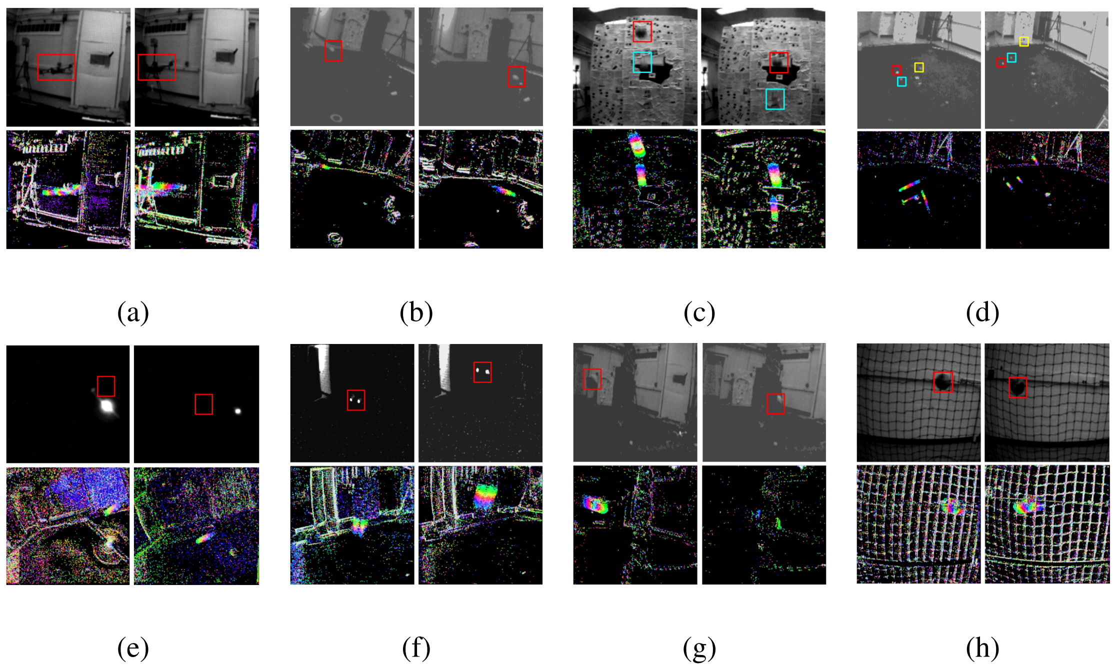


Figure 3.9: *Sample frames from the EED dataset are shown. Top row shows RGB frames with superimposed bounding boxes for tracked object. The bottom row displays the corresponding time-images. (a) - Remote Controlled Quadrotor recording, (b), (c) and (d) - Single, two and three simultaneously moving objects, (e) - Strobe Light sequence, (f) - low lighting (no strobe), (g) - An "Occluded Sequence", (h) - "What is a Background?" sequence*

Chapter 4: Learning Sensori-Motor Control with Neuromorphic Sensors Using Hyperdimensional Binary Vectors

4.1 Methods

As was shown in Chapter 3, the temporal information contained within Time images is complimentary to scene optical flow and can directly be used in motion estimation tasks. Here we use a theory developed by Kanerva [54], and which has been most often used for text processing, to develop a learning system which solves an egomotion estimation task with event-based sensor.

The essence of the Kanerva's theory, is that if the information can be presented via a sufficiently long binary vector such that the statistical distribution of bits within this vector correlates with the desired output, it is possible to construct a system capable of learning on very small data samples (500 frames in our experiments) in a single pass, and performing inference at extremely high rates. We delve deeper in the theory behind our method in Appendix B; for now we just outline our encoding scheme and findings. Similar to Chapter 3, we start by constructing Time images; we then compute the 4-parameter model using Gray codes - to ensure that similar numbers are close in Hamming space and use the resulting encodings to build a Hyperdimensional Binary Vector (HBV).

During the learning stage, we cluster images with similar velocities (along each degree of freedom separately) - $a...z$ below - and combine them using Consensus Sum (here $'+'$) and XOR (shown as multiplication). The equivalent of the trained model in HBVs is called a 'memory':

$$m = v_1(a_1 + a_2 + ...) + v_2(b_1 + b_2 + ...) + ... + v_n(z_1 + z_2 + ...) \tag{4.1}$$

Here, $v_1..v_n$ are randomly generated 'basis' vectors which can be then used to separate encoded motion classes from each other. This model is used to generalize and extract the egomotion in a self driving car scenario.

4.2 Implementation

To perform our experiments, we have developed an open-source library for Python for accelerated manipulations with long binary vectors (pyhdc). The library supports vector lengths of up to 8000 bits and is capable of performing $1.4 * 10^5$ permutations per second and $3.0 * 10^7$ XORs per second on a modern laptop in a single thread. The code is available online: https://github.com/ncos/pyhdc.

4.3 Results

We want to emphasize the ability of our pipeline to train in a single pass in our experiments, with only 500 samples, taken from outdoor day 1, while producing comparable results on this and other sequences. Neural approaches tend to use a much higher ratio of training to test set. We present the qualitative results of our ego-motion estimation in Fig. 4.1 on outdoor day 1, as well as the other subsets. We also provide quantitative results in Table 4.1. The binning of velocity sizes is directly proportional to our inference time. For 500 velocity classes, we perform at an average of 572 inferences per second (time slices per second). If we halve the number of velocity classes (250), then the inference rate approximately doubles (1084). Training time is extremely fast and proportional to the number of training frames, requiring only 0.5 and 2 s for 500 and 1500 frames, respectively. Retraining on the fly is possible, and training can be done in an online fashion. We received these results on a regular laptop central processing unit, running at 3 GHz.

Table 4.1: *Quantitative results for the MVSEC dataset across the five subsets. The table gives the number of frames in each subset, length in seconds, the size of the angular and linear bins used for HBVs, number of vectors in the rotation, X and Z, and the Average Endpoint Error (AEE), average relative pose error (ARPE), and the average relative rotational error (ARRE).*

MVSEC subset	**Outdoor day 1**	**Outdoor day 2**	**Outdoor night 1**	**Outdoor night 2**	**Outdoor night 3**
Frames	5134	12196	5133	5497	5429
Length (s)	128.3	304.9	128.3	137.4	135.7
Ang. bin (rad/s)	0.02	0.02	0.02	0.02	0.02
Lin. bin (m/s)	0.08	0.08	0.08	0.08	0.08
Rotation (clusters)	104	101	40	74	87
X (left-right, clusters)	47	44	24	40	33
Z (front-back, clusters)	119	311	244	251	228
AEE	0.810	1.03	0.933	1.16	0.940
ARPE	0.122	0.225	0.243	0.0948	0.0831
ARRE	0.0994	0.108	0.0632	0.116	0.121

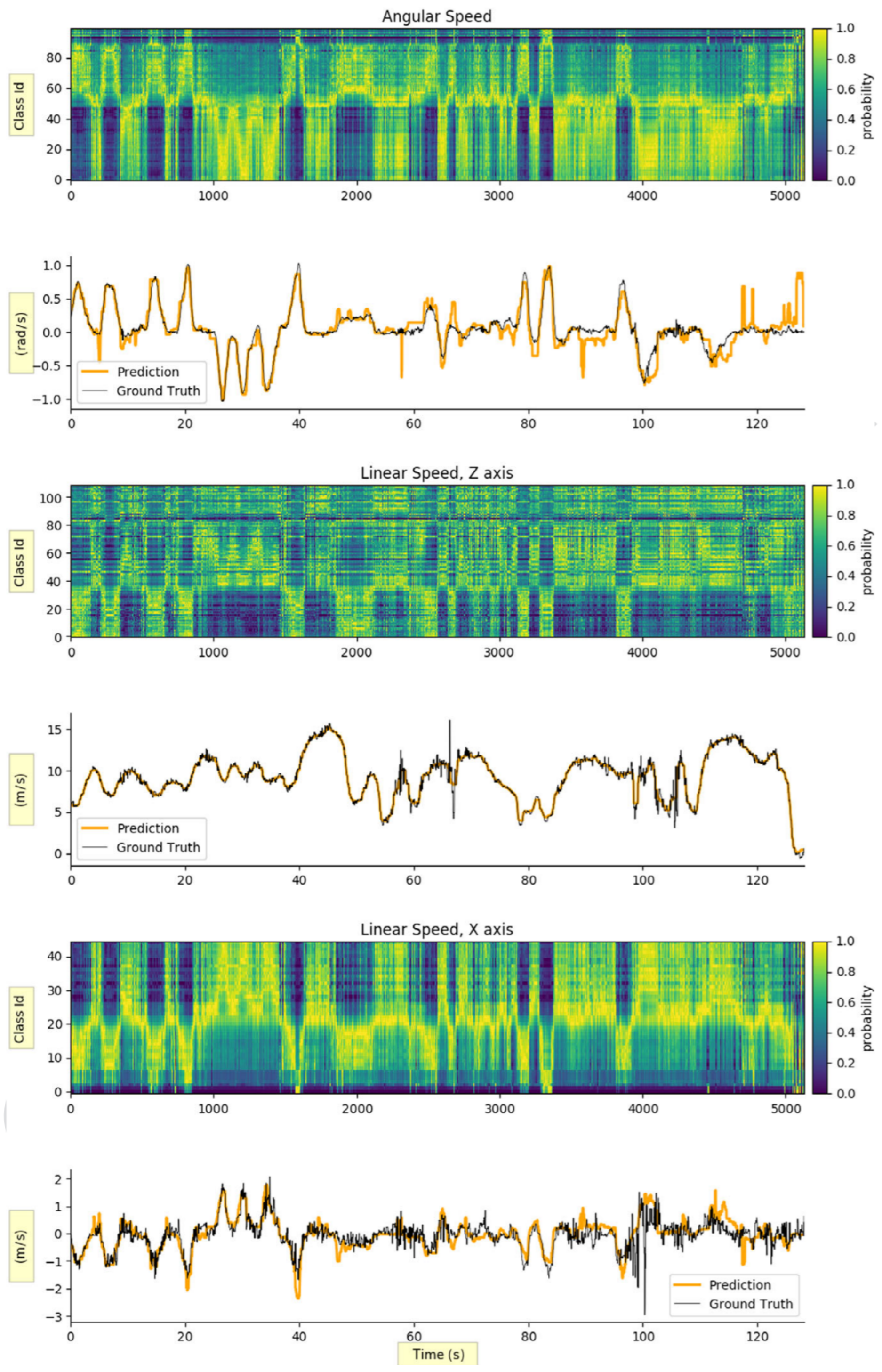


Figure 4.1: *Results for HBVs on outdoor day 1 in the MVSEC dataset. Qualitative results on the outdoor day 1 subset of the MVSEC dataset. The probability plots show the likelihood of each velocity class being correct across all inferences, with a light-green color indicating the higher probabilities. One can see that the light green forms the path of predictions across the probability field.*

Chapter 5: EVIMO: A Dataset for Motion Segmentation with Event-Based Cameras

5.1 Introduction

EVIMO is a collection of indoor datasets for motion segmentation and egomotion estimation gathered with a variety of event-based sensors (currently, only monocular DAVIS346C dataset is available, but the list of recordings is being expanded). The dataset features 6 DoF poses for Camera and Independently Moving Objects updated at 200 Hz, and pixelwise object masks at 40 Hz. Due to the way the data was collected, the object masks can be re-generated at any frame rate up to 200 Hz, or higher with pose interpolation. Depth maps and classical camera frames are also available for most sequences. We support Python (numpy) .npz format, as well as plain text format. We also provide raw ROS .bag recordings and C++ code used to process the recordings.

We release the dataset and raw recordings here: ***https://better-flow.github.io/evimo/index.html***
The code for the labeling pipeline can be found here: ***https://github.com/better-flow/evimo***

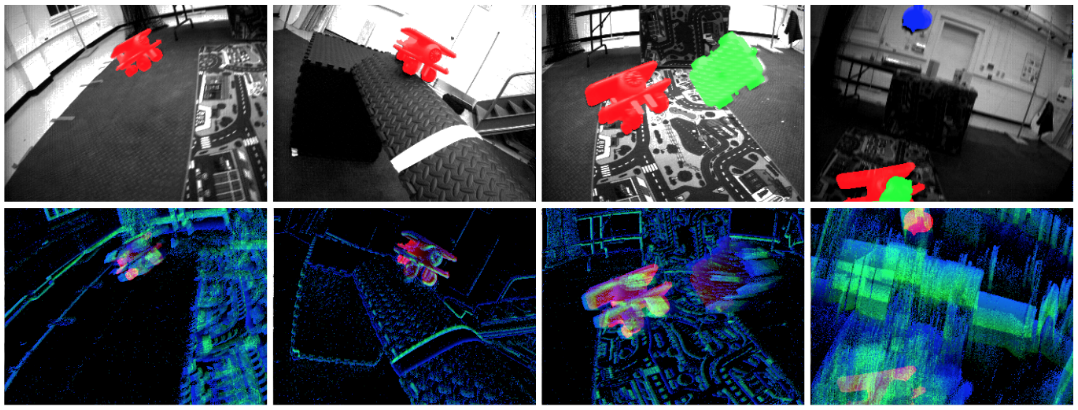


Figure 5.1: *An example from EVIMO dataset. Motion segmentation masks can be provided at up to 200 frames per second (or higher, with motion interpolation). The dataset is annotated automatically by combining motion capture data with static scans of the environment and objects.*

5.2 EV-IMO Dataset

Event-based sensors essentially do not suffer from motion blur, and are capable of working in poor light conditions where classical cameras fail. To provide the best possible ground truth, we use Vicon motion capture system to track the locations of the camera and objects. The ground truth depth and masks are then synthesized offline during data postprocessing by projecting static 3D scans of the room and the objects onto camera plane. This approach allows to capture data in extreme dark and avoid motion blur for fast moving objects, without sacrificing the quality of the dataset. *EV-IMO* dataset is the first event camera dataset to include multiple independently moving objects and camera motion (at high speed motion), while providing accurate depth maps, per-object masks and trajectories at over 200 frames per second. We used this dataset in [31] and Chapter 6.

5.2.0.1 Methodology

Event cameras such as the DAVIS are designed to capture high speed motion and work in difficult lighting conditions. For such conditions classical methods of collecting depth ground truth, by calibrating a depth sensor with the camera, are extremely hard to apply - the motion blur from the fast motion would render such ground truth unreliable. Depth sensors have severe limitations in their frame rate as well. Furthermore it would be impossible to automatically acquire object masks - manual (or semi-automatic) annotation would be necessary. To circumvent these issues we designed a new approach (Fig. 5.2 demonstrates the interface of the auto annotation tool):

1. A static high resolution 3D scan of the objects, as well as 3D room reconstruction is performed before the dataset recording takes place.

2. The Vicon® motion capture system is used to track both the objects and the camera during the recording.

3. The camera center as well as the object and room scans are calibrated with respect to the the Vicon coordinate frame.

4. For every pose update from the Vicon motion capture, the 3D point clouds are transformed and projected on the camera plane, generating the per-pixel mask and ground truth depth.

This method allows to record accurate depth at very high frame rate, avoiding the problems induced by frame-based collection techniques. While we acknowledge that

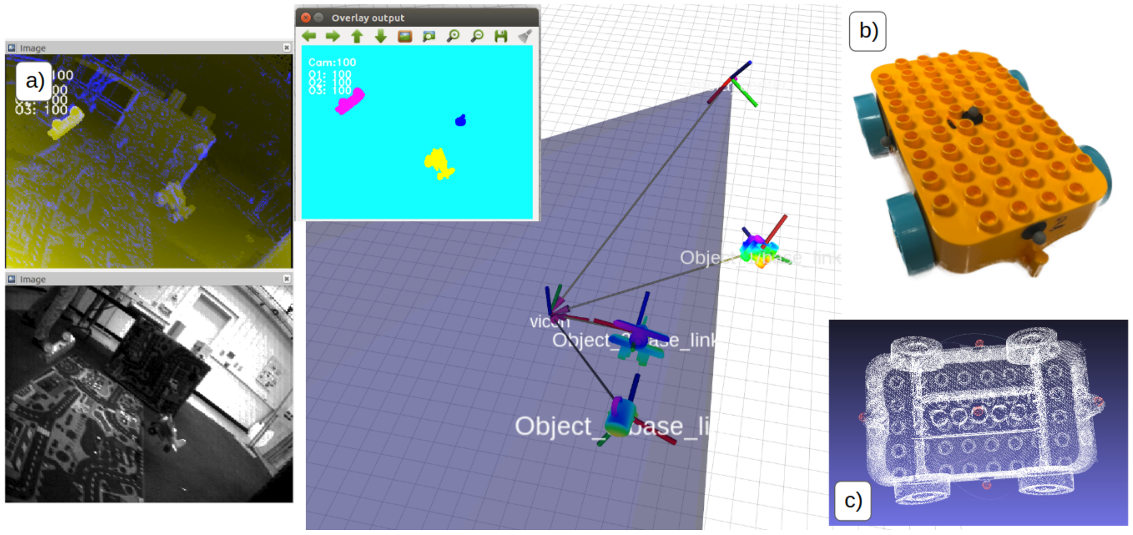


Figure 5.2: *a) - The main interface of the automatic annotation tool. Camera cone of vision, depth and motion masks are visible. b) - Example object used in the dataset. c) 3D scan of the object.*

this approach requires expensive equipment, we argue that our method is superior for event-based sensors, since it allows to acquire the ground truth at virtually any event time stamp (by interpolating poses provided at 200 Hz) - something impossible to achieve with manual annotation.

5.2.0.2 Dataset Generation

Each of the candidate objects (up to 3 were recorded) were fitted with Vicon® motion capture reflective markers and 3D scanned using an industrial high quality 3D scanner. We use RANSAC to locate marker positions in the point cloud frame and using acquired point correspondences we transform the point cloud to the world frame at every update of the Vicon. To scan the room, we place reflective markers on the Asus Xtion

RGB-D sensor and use the tracking as an initialization for global ICP alignment.

To compute the position of the DAVIS camera center in the world frame we follow a simple calibration procedure, using a wand that is tracked by both Vicon and camera. The calibration recordings will be provided with the dataset. The static pointcloud is then projected to the pixel coordinates (x, y) in the camera center frame following equation 5.1:

$$(x, y, 1)^T = KCP_{davis}^{-1}P_{cloud}X_i \tag{5.1}$$

Here, K is the camera matrix, P_{davis} is a 4×4 transformation matrix between reflective markers on the DAVIS camera and the world, C is the transformation between reflective markers on the DAVIS and DAVIS camera center, P_{cloud} is the transformation between markers in the 3D pointcloud and reflective markers in the world coordinate frame, and X_i is the point in the 3D scan of the object.

Or dataset provides high resolution depth, pixel-wise object masks and accurate camera and object trajectories. We additionally compute, for every depth ground truth frame, the instantaneous camera velocity and the per-object velocity in the camera frame, which we use in our evaluations. We would like to mention, that our dataset allows to set varying ground truth frame rate - in all our experiments we generated ground truth at 40 frames per second.

5.3 Sequences

A short qualitative description of the sequences is given in Table 5.1. We recorded 6 sets, each consisting of 3 to 19 sequences. The sets differ in the background (in both depth and the amount of texture), the number of moving objects, motion speeds and lighting conditions.

A note on the dataset diversity: it is important to note, that for event-based cameras (which capture only edge information of the scene) the most important factor of diversity is the variability on *motion*. Different motions create 3D event clouds which vary significantly in their structure, even with similar backgrounds. Nevertheless, we organize our sequences into four background groups - *'table'*, *'boxes'*, *'plain wall'* and *'floor'* (see Fig 5.3), with the latter two having varying amounts of texture - an important factor for event cameras. We also include several tabletop scenes, with clutter and independently moving objects.

5.4 Results on Learning Pipelines

Here we provide some results on motion estimation, segmentation and depth estimation from our previous work [31] - a convolutional neural network based on SfM-Learner. A dual neural network consists of separate depth and pose networks, jointly trained using both ground truth and a warping loss.

A depth network uses an encoder-decoder architecture and is trained in supervised mode to estimate scene depth. A pose network takes consecutive event slices to generate

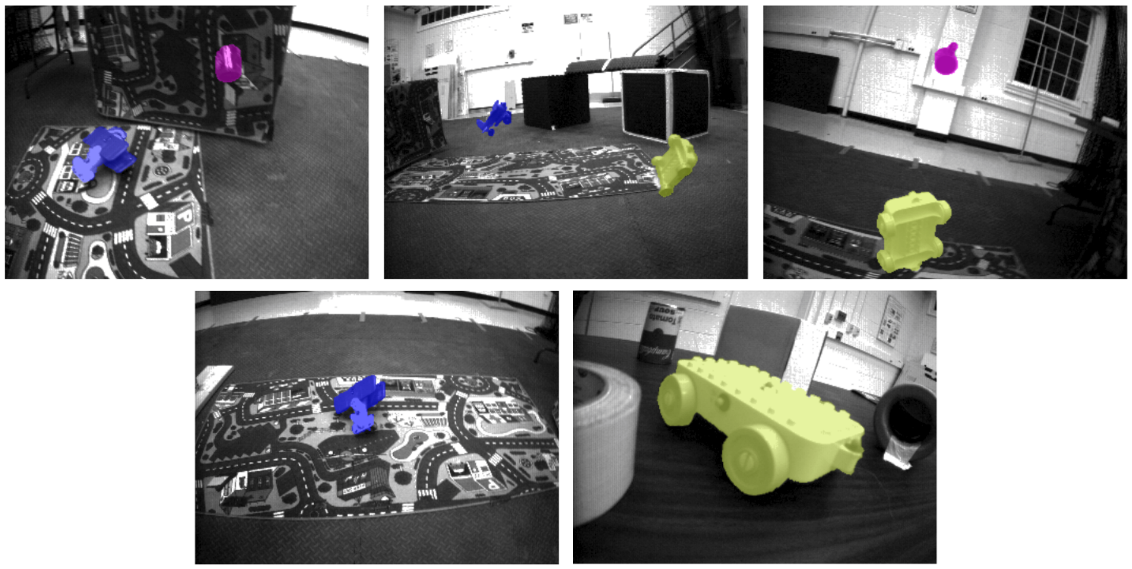


Figure 5.3: *Types of background geometry featured in the EV-IMO dataset (from left to right): 'table', 'boxes', 'plain wall', 'floor' and 'tabletop'.*

a mixture model for the pixel-wise pose. The outputs of the two networks are combined to generate the optical flow, then to inversely warp the inputs and backpropagate the error - to provide dense depth, egomotion, pixelwise motion for independently moving objects and motion masks.

Please refer to Chapter 6 and Appendix A for additional results on segmentation in (x, y, t) space.

5.4.1 Qualitative Results

We present a qualitative evaluation in Fig. 5.4. The per-object pose visualization (Fig. 5.4, columns 4 and 5) directly maps the 3D linear velocity to RGB color space. The network [31] is capable of predicting masks and pixel-wise pose in scenes with different amount of motion, number of objects or texture.

	background	speed	texture	occlusions	objects	light
Set 1	boxes	low	medium	low	1-2	normal
Set 2	floor/wall	low	low	low	1-3	normal
Set 3	table	high	high	medium	2-3	normal
Set 4	tabletop	low	high	high	1	normal
Set 5	tabletop	medium	high	high	2	normal
Set 6	boxes	high	medium	low	1-3	dark / flicker

Table 5.1: *EV-IMO sequences*

5.4.2 Motion Estimation and Segmentation

To evaluate the linear components of the velocities, for both egomotion and object motion, we compute the classical Average Endpoint Error (AEE). Since our pipeline is monocular, we apply the scale from the ground truth data in all our evaluations. To account for the rotational error of the camera (which does not need scaling) we compute the Average Relative Rotation Error $RRE = \|logm(R_{pred}^T R_{gt})\|_2$. Here $logm$ is the matrix logarithm, and R are Euler rotation matrices. The RRE essentially amounts to the total 3-dimensional angular rotation error of the camera. We also extract several sequences featuring fast camera motion and evaluate them separately. We present AEE in m/s, and $ARRE$ in radians/s in Table 5.2.

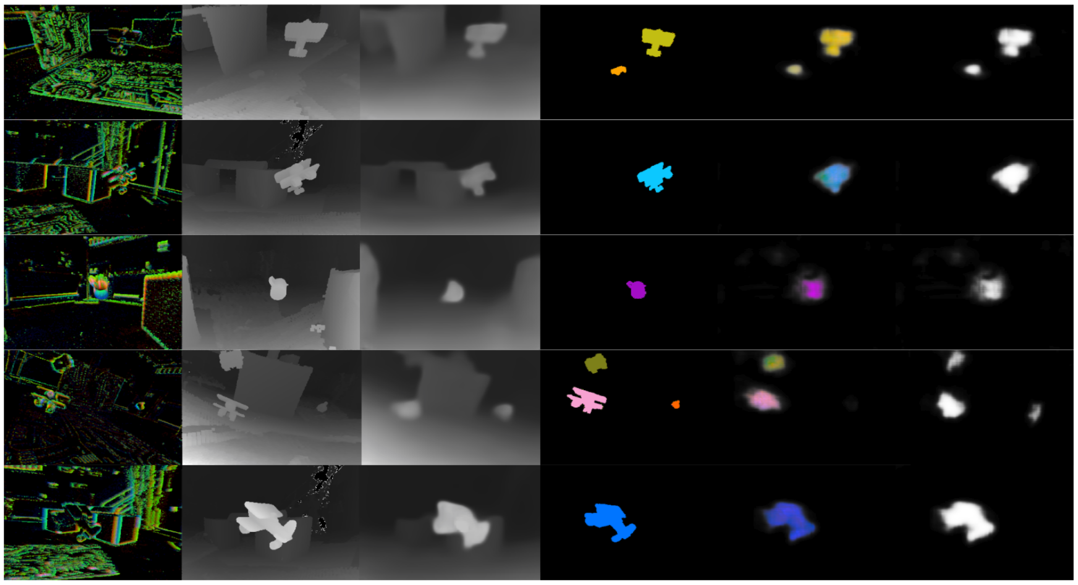


Figure 5.4: *Qualitative results from our evaluation. The table entries from left to right: DVS input, ground truth for depth, network output for depth, ground truth pixel-wise pose, predicted pixel-wise pose, predicted motion mask. Examples were collected from EV-IMO dataset. Best viewed in color.*

5.4.3 Depth Estimation

Both egomotion and depth estimation is significantly harder with complex 3D motion. We evaluate [31] against a recent method [27] - *ECN* network, which estimates optical flow and depth on the event-based camera output. The method was originally designed and evaluated on a road driving sequence (which features a notably more simple and static environment, as well as significantly rudimentary egomotion). Still, we were able to tune [27] and train it on *EV-IMO*. We provide the comparison for the depth for our baseline method, the smaller version of our network (with just 40k parameters) and *ECN* in Table 5.3.

	Cam AEE	Cam $ARRE$	Obj AEE	IOU
$table$	0.07 (0.09)	0.05 (0.08)	0.19	0.83 (0.63)
$plainwall$	0.17 (0.23)	0.16 (0.24)	0.38	0.75 (0.58)
$fastmotion$	0.23 (0.28)	0.20 (0.26)	0.43	0.73 (0.59)

Table 5.2: *Evaluation on segmentation and motion estimation. The numbers in braces are values for the 40k version of the network. AEE is in m/s, $ARRE$ is in rad/s.*

	Error metric			Accuracy metric		
	Abs Rel	RMSE log	SILog	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Baseline Approach						
plain wall	0.16	0.26	0.07	0.87	0.95	0.97
cube background	0.13	0.20	0.04	0.87	0.97	0.99
table background	0.31	0.32	0.12	0.74	0.90	0.95
ECN						
plain wall	0.67	0.59	0.33	0.27	0.52	0.80
cube background	0.60	0.56	0.30	0.29	0.53	0.78
table background	0.47	0.48	0.23	0.45	0.69	0.86

Table 5.3: *Evaluation of the depth estimation*

Chapter 6: Learning motion segmentation in (x, y, t) space using edge convolutional neural networks

6.1 Introduction

Scene motion *exists in time*. Changes in the scene over short time interval provide some information, but tasks, such as occlusion detection, segmentation of multiple moving objects, and detecting objects with a motion similar to the camera, require capturing the changes over longer time-intervals to resolve ambiguities. In classical vision, feature point tracking allows for long-term estimation of pixel motion trajectories. That way, by analyzing pixel matches over large time intervals the ambiguity in motion can be resolved.

Classic frame-based vision, however, is not naturally designed to provide temporal information. The technical trends to increase the image resolution is well aligned with the goals of static visual challenges like recognition or semantic segmentation. The appearance-based algorithms, which attempt to analyze how the world around *is*, are not naturally designed to provide temporal information; this can be in part due to past technical trends which shaped the evolution of a modern camera. The possibility to instantaneously capture a scene in high detail was indeed very appealing and not without merit - it is well aligned with the goals of static visual challenges like recognition or semantic

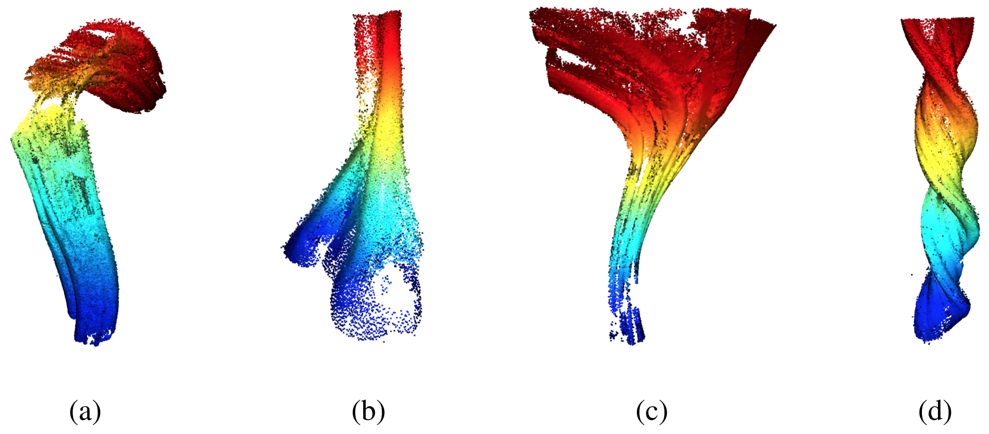


Figure 6.1: *Illustration of event cloud shape properties depending on camera or object motion. Each point is an event, the timestamp is shown with color: blue - 0 sec, red - 1 sec. (a) - translation parallel to camera plane (along y axis), (b) - roll (rotation around y axis), (c) - translation along z axis (object moves closer to camera), (d) - yaw (rotation around z axis)*

segmentation. The works pursuing to extract scene motion, on the other hand, had to make use of the same sensors and rely on artificially crafted approaches tailored to work with static data: feature points, and the concept of *matching* is one of them; this idea to use appearance of the scene to extract object motion is by itself forced by the nature of classical cameras.

Examples of event-based input are shown in Fig. 6.1. For four object motions, the event clouds are illustrated (the color of points corresponds to the event timestamp, in the range of 0 to 1 sec.): (a) translation does not change the shape of the object across time, only its spatial coordinate; (b) roll, or rotation around the axis parallel to the camera plane reveals previously occluded parts of the object, and the shape of the cloud cross-section changes. (c) With translation along the camera's optical axis, as the object approaches the camera its contour becomes larger. (d) Rotation around optical axis produces a very discernible twist pattern. In addition, it is easy to see how natural the object tracking problem becomes, given the high temporal density of events. We conduct formal analysis of 3D motion pattern features in Sec. 6.2.

The time-surfaces have their origin in the Epipolar plane image analysis introduced in the late 80's [55]. As the camera moves along a linear path, images were taken in such rapid succession that they formed a solid block of data. The technique utilized knowledge of the camera motion to form and analyze slices of this solid. These slices directly encode not only the three-dimensional positions of objects, but also spatiotemporal events such as the occlusion of one object by another. For straight-line camera motions, these slices have a simple linear structure that makes them easy to analyze.

Generalizing this concept, we work with time-surfaces in (x, y, t) produced by an event camera to perform the task of foreground-background segmentation of moving objects. The complexity of this task comes from the enormous amount of asynchronous data which needs to be processed, high levels of noise produced by the event camera and, most importantly, the fact that *the variability of object and camera motion changes the shape of the event cloud*, making it more difficult to learn local 3D features. In summary, our contributions are:

- The first learning approach capable of working on 3D event clouds over large time intervals.
- We show theoretically and with experiments that larger temporal slices yield better performance.
- We perform comparisons to current state of the art, using *PointNet++* [56] and

EVIMO [31] as baselines, and show that our method is faster and yields better results.

6.2 Event Clouds and Scene Motion

Event cameras record a continuous stream of brightness change events. Each event is encoded by its pixel position x, y, timestamp, t, accurate to microseconds, and an additional bit denoting whether brightness increased or decreased. We will refer to the events in a fixed-time interval as 'slice' of events. A single slice can contain millions of events (practically, 10^6 per second on EVIMO [31] dataset collected with DAVIS 346C, and $6 * 10^6$ for a newer PROPHESEE high resolution sensor [3]).

The events are generated by the motion of image contours (object boundaries, or texture edges on objects) over time. We can look at these events as points on trajectories $tj(t) = (x(t), y(t), t)$ - this makes it natural to look at events as points in 3D (x, y, t) space. The rest of this section is devoted to analyzing some geometric properties of the event clouds. Since pixel motion is constrained by the laws of physics, the rigidity of bodies and epipolar geometry, event clouds are significantly different from 3D pointclouds in (x, y, z) space.

6.2.1 Surface Normals, Normal Flow, & Optical Flow

Earlier works [13, 14] have analyzed surface normals of event clouds in the context of optical flow estimation. With some abuse of notation, similar as in [13], we describe the event cloud as a function $t(x, y)$, with t the time and x, y the spatial coordinates,

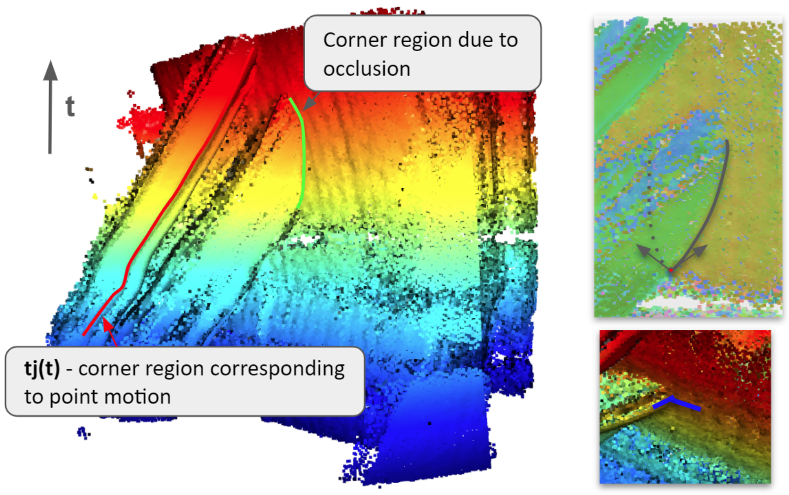

t
Corner region due to occlusion
tj(t) - corner region corresponding to point motion

Figure 6.2: *Event cloud in (x,y,t) space; the color gradient corresponds to event timestamps. For a single rigid object, curvatures on the cloud would define a trajectory of a point of this object, while for a pair of objects such structure can identify an occlusion. On the top right - a point with two possible optical flow values, which only occur during occlusions. Bottom right: T-corner; during occlusion its shape defines which object is on the foreground and which is on the background (see Sec. 6.2.3)*

and consider it a surface (actually multiple x, y may map to the same t violating the definition of a function). Then the partial derivates $\frac{\partial t(x,y)}{\partial x}$, $\frac{\partial t(x,y)}{\partial y}$ provide the inverse of the observed velocity components. Thus, from the normal to this surface n with components (n_x, n_y, n_t), the *instantaneous normal flow* at an image pixel can be obtained as $v_n = (\frac{n_t}{n_x}, \frac{n_t}{n_y})$.

More importantly though is the notion that the full flow of a point $p = (x, y, t)$ would lie in the plane with the normal n and passing through p. Now, we can impose an additional assumption, which is often called *smoothness constraint* - that flow in the local

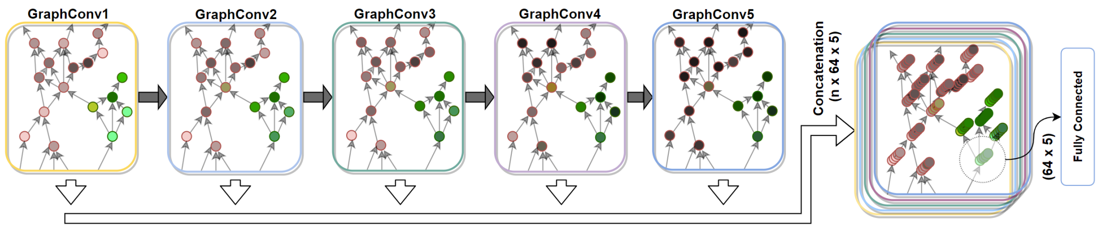


Figure 6.3: *An illustration of the architecture. 5 Graph Convolutional layers aggregate multiscale features around each point; the features are concatenated and fed into a fully connected layer to predict point class.*

region is similar; this is not true for the boundaries, and we analyze boundary regions separately below.

More specifically, for a region of a small radius r around the point $p_0 = (x, y, t)$ we will assume that all points $p_i \in B_r(p_0)$ have the same (normalized) optical flow vector $v = (v_x, v_y, v_t)$. This results in a constraint: *the optical flow at* p_0 *lies at the intersection of all local planes given by normals* n_i *and passing through* p_0. If the assumption holds true, and flow is the same for all local points, then all planes n_i will intersect on a single line v. The intersection of two planes given by n_i and n_j is simply $v = n_i \times n_j$. Given a set of planes in $B_r(p_0)$, we formulate the constraint on flow as a least squares problem:

$$v = min_v \sum ||v \times n_i||^2 \tag{6.1}$$

Here, we deviate from the common notion of optical flow as a $2D$ vector field; given the frame-less nature of the event stream, v should rather be thought of as the temporal derivative of the pixel trajectory, while instantaneous flow in a traditional sense can be written as $v_{xy} = (\frac{v_x}{v_t}, \frac{v_y}{v_t})$.

6.2.2 Continuity in Time

The flow is the derivative of the point trajectory $v = \frac{d(tj)}{dt} = (\frac{dx}{dt}, \frac{dy}{dt}, 1)$. Due to the *continuity in time* of the event stream, sometimes it is possible to recover the full trajectory of a point. A trajectory is unambiguously recoverable if all its points are *generalized corners* - that is, for all points Eq. 6.1 needs to have a single minima. Then, using a sequence of candidate points p_i and their corresponding flow v_i, it is possible to extract the full trajectories of these points. Even, if the data is discontinous, we may be able to obtain the trajectory. **Note**, a property of event camera is that it does not generate events when there is lack of relative motion. In such case, at a point p_i (given the relative motion, or flow is zero), $v_i = (0, 0, 1)$ will point vertically upwards, and since there was no motion, the point p_{i+1} will be directly above p_i, allowing to recover from the gap in the event stream.

6.2.3 Boundary Regions and Occlusions

Corner detection for event cameras has been studied previously [23]. However, existing methods don't allow to distinguish between object corners, and structures caused by occlusions (see Fig. 6.2). Next we discuss how to distinguish these cases.

A consequence of Eq. 6.1 is that every point with nonzero curvature will have an unambiguous optical flow vector associated with it. If a point region $B_r(p_0)$ includes events from boundaries of two separate objects (which can happen during occlusions) Eq. 6.1 will have multiple distinct minima. An example is shown in Fig. 6.2: on the top - a typical occlusion of the background (shown as large, flat cluster of points) by a smaller foreground object (shown as a ’tube’ cutting through the background motion plane).

To *distinguish* between a corner corresponding to a point trajectory (shown in red in Fig. 6.2) and a corner caused by the occlusion (shown in green), it is enough to analyze the local distribution of the optical flow vectors. On the bottom left - the red point has multiple possible flow vectors, and its trajectory $\frac{d(tj)}{dt} = \infty$. These special points can be found and corresponding corner regions labeled as *occlusion boundaries*.

It is also possible to extract at the occlusion, information about which of the two object is in the foreground, and which is in the background. Since due to occlusion the texture on the occluded object is not visible, the foreground edge surface will be *hollow* inside and the background surface will be intersected by the foreground one. On the occlusion boundary this will always result in a T-shaped corner, shown on Fig. 6.2, bottom right.

6.3 The Architecture

6.3.1 Motion Segmentation in 3D

State-of-the-art 3D point cloud segmentation networks such as PointNet++ [56], EdgeConv [57], and 3DCNN [58] are tailored to extract static 3D feature descriptors in a uniform 3D metric space. The event cloud differs in that it has a temporal axis; the motion of the object itself controls the shape of the cloud and hence static (x, y, t) features cannot be learnt as descriptors. The metric space also allows for efficient downsampling and mesh simplifications, with modern sensors having notably less noise as opposed to event cameras - hence only a handful of methods is capable of handling millions of points produced by event cameras. In this work we use PointNet++ as a baseline and develop network based on Graph Convolutional Networks to perform the task of background-foreground motion segmentation.

6.3.2 Network Design

Our network architecture is shown in Figure 6.3 - it consists of five consecutive Graph Convolutional (GConv) layers and three fully connected hidden layers which share the same weights across all points and perform global feature aggregation. The input to the network is an unstructured graph which consists of events in (x, y, t) space as nodes (combined with per-point surface normals (n_x, n_y, n_t)) and a set of edges computed as described in Sec. 6.3.3.

In each GConv layer, every point feature is aggregated from its neighbours, making points with similar vertices clustered together. When training multiple GConv layers, it is equivalent to a multi-scale clustering of the point features that also preserves local geometric structures [59]. Each GConv layer has 64 input channels and maps the features to 64 output channels at different scales across the entire graph [59]. Then, the 5 sets of 64 multi-scale features extracted by the five GConv layers are concatenated and fed to a MLP classifier. The MLP starts with 256 initial channels and reduces the channels by a rate of 4 into 16 channels in the last hidden layer. Then, the outputs of the MLP are connected to a fully connected layer to merge into a single point-wise score.

We supervise our training as a regression problem instead of a classification problem to decrease the possibility of overfiting to object contours. The raw response values are compared with the point-wise ground truth labels $\in [0, 1]$ by Binary Cross Entropy with Logits Loss.

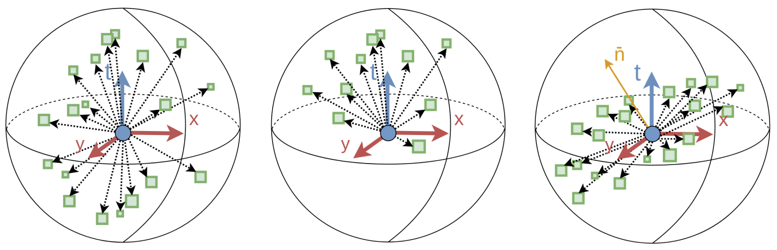
6.3.3 Edge Computation

A central concept for Graph Convolutional neural networks is edge computation strategy. A high density of points in event cloud could cause exponential grow in the number of edges, making large edge radii prohibitively expensive to compute and use. To get around this limitation, while preserving connections with neighbouring points along possible motion trajectories, we filter the edges in a sphere radius r so that they are parallel to event-surface. The Eq. 6.3-(a) along with Fig. 6.4-(b) signifies this relation: given a point p_0 and its normal n_0, we would keep only the edges orthogonal to n_0, with a certain filtering threshold α. Most of the temporal motion information of an event is contained within a plane parallel to time-surface, and this filtering strategy is a good trade-off between the richness of surface features and computational performance. Yet, our experiments have shown that most surface patches are rather isolated in the absence of strong texture or extremely fast motion, and in practice no filtering is required to produce edge pattern following Eq. 6.3(a).

We notice that for a given point it is enough to consider just the points in the upper (along temporal axis) hemisphere - this halves the number of edges with little to no decrease in network performance. Essentially, we combine Eq. 6.3(a) and (b), to produce sparse edges which will maximally *align with possible local optical flow values* (see Sec. 6.2). In our experiments we use $r = 10$ pixels (scaling of temporal axis is described in Sec. 6.4.1.1) and we perform an ablation study on various radii, shown on Fig. 6.7(b).

$$(a) : \{p_i | \forall p_i \in B_r(p_0) \quad \& \quad (p_i - p_0) \cdot n_0 < \alpha\} \tag{6.2}$$

$$(b) : \{p_i | \forall p_i \in B_r(p_0) \quad \& \quad p_i^t > p_0^t\} \tag{6.3}$$



(a) (b) (c)

Figure 6.4: *Edge configurations: (a) - all points are used to create edges - a typical configuration used in 3D processing; (b) - only points in the upper hemisphere are used (we use this configuration in all our experiments); (c) - edges are parallel to time-surface.*

6.3.4 Temporal Augmentation

Following intuitions in Sec. 6.2, we propose *temporal augmentation* to artificially introduce variations in object speed (and hence - cloud shape) and improve generalization on varying motion. Given a point in an input cloud $p = (x, y, t)$ with a normal $n = (n_x, n_y, n_t)$ the augmented point p' and its normal n' are computed as:

$$p' = (x, y, \alpha * t) \quad n' = (\alpha * n_x, \alpha * n_y, n_t) \tag{6.4}$$

Where α is a random scaling parameter (in our work, $0.8 - 1.2$) and n' is normalized to a unit vector length.

The motivation of this method is that the cues which signify a separate object do not depend on its (or camera) speed, but rather on spatio-temporal *anomalies* - such as T-corners, points with multiple flow values or continuity of event cloud; this augmentation forces the neural network to learn such features (which remain unaffected by augmentation), and reduce overfitting to the local shape of the cloud. We perform ablation studies in Sec. 6.4.3 and find that the model generalizes better when this augmentation is used. The possibility of performing the temporal augmentation is a major advantage of learning on 3D event clouds and would be not feasible in metric (x, y, z) space.

6.4 Experiments

6.4.1 Dataset

EVIMO is the only publicly available event-based segmentation dataset. It includes pixelwise masks at 200Hz and roughly 30 minutes of recording, although the dataset was collected in a single room and only includes three objects, so segmentation methods might be prone to overfitting. To alleviate these issues, our pipeline is not trained on depth so overfitting to room structure is unlikely. The 3D shape of the objects varies a lot, depending on object motion, but overfitting on contours is still possible. We perform an ablation study in the supplementary material, where we train a NN using only sequences with 2 of the objects and evaluate on sequences with the third object.

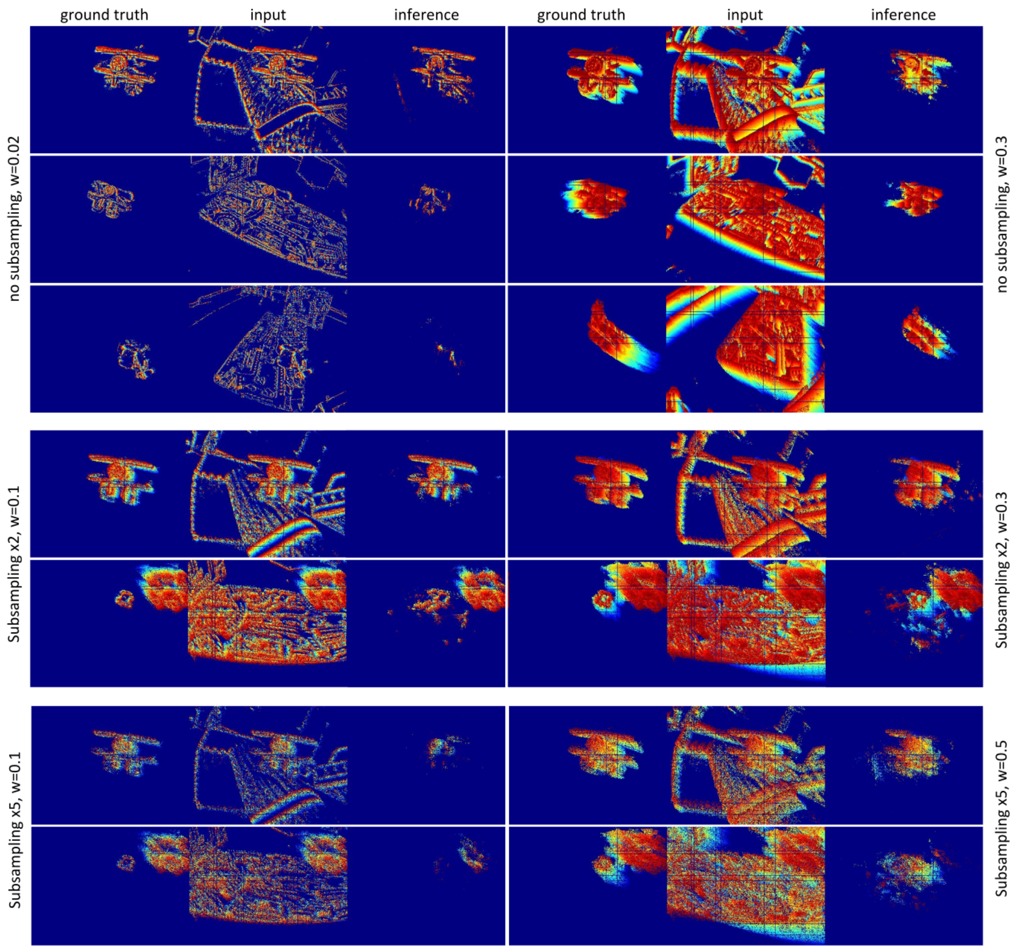


Figure 6.5: *Qualitative results - the event cloud was downprojected, color represents the normalized timestamp. The subsampling rate increases from top to bottom; small time slices are shown on the left, and large on the right. Experiments with* 0.5 *second slices were only possible to achieve with* $u = 5$ *uniform downsampling factor, but the dramatic decrease in feature quality yields poor results compared to* 0.3 *second slice with no subsampling. Note how the results improve for larger slice width, especially in the presence of texture.*

6.4.1.1 Data Preprocessing

We preprocess the raw data provided in EVIMO. We upscale the temporal axis of the event cloud by a factor of 200 to keep the density of events more uniform across the x,y,t axes. The normals are precomputed using PCA, with $r = 5$ (after temporal upscaling). We also apply a radius outlier filter with $r = 3$ and $k = 30$ - which removes all points having less than 30 points in a radius of 3 pixels; this results in approximately 10% to 15% points removed. To further reduce the number of points, we use random subsampling with a factor u, which keeps 1 random point out of u (in practice, we use $u = 1$ for no subsampling or $u = 2$, to remove half the points). We precompute edges within a radius of 10 pixels for all points, and (optionally) keep up to 30 edges connected to nearest points.

The ground truth provided in EVIMO is image-based, sampled at $200Hz$. For every event, we approximate its class by locating the nearest class mask according to mask and event timestamps and downprojecting the event coordinate onto the image. This produces good results in practice, even for fast motion; we show a sample of the event cloud with class annotations in the supplementary material.

6.4.2 Implementation Details

Our network architecture consists of five consecutive Graph Convolutional(GC) layers and three fully connected hidden layers. As input features we use spatio-temporal event locations (x, y, t), time-surface normals (n_x, n_y, n_t), and a per-event binary polarity value ($p \in \begin{bmatrix}0, & 1\end{bmatrix}$) which signifies whether the brightness has increased or decreased at pixel (x, y), at time t.

In our current implementation (6.3.2), the network has $117k$ trainable parameters. We train the network with a batch size of 3 using three Nvidia GTX 1080Ti GPUs. Larger batch sizes are difficult to achieve in practice because the size of time slices is very high. We use the Adam optimizer with a learning rate of $7e-4$ and cosine annealing scheduling strategy. We train our models for 200 epochs on a full train section of $EVIMO$ [31] dataset. The model takes 6-minutes to train for each epoch with a slice width of 0.3 seconds and no subsampling.

As baseline we use the state-of-the-art *PointNet++* [56] on event clouds. We take the segmentation implementation of *PointNet++* presented in the paper [56] which inspired by and further improved results on 3DCNN [58] and PointNet [60]. The implementation consists of two hierarchical sampling & grouping modules, followed by one k-nearest-neighbours interpolation and three forward passing modules. K-nearest-neighbours are searched with a ratio and a radius of $0.2/0.2$ in the first sampling layer and $0.25/0.4$ in the second sampling layer. The responses from the last forward passing modules are passed into three concatenated layers of fully connected network to produce a one label output for each point.

6.4.3 Ablation Studies

6.4.3.1 Effects of the Slice Width

We conduct an experiment to investigate the effects of slice width and cloud subsampling on the performance of the network. Fig. 6.6 shows results on *boxes* valida-

	boxes			floor			wall			table			fast		
	0.3	0.1	0.02	0.3	0.1	0.02	0.3	0.1	0.02	0.3	0.1	0.02	0.3	0.1	0.02
$GConv$	84±9	81±8	60±18	80±9	79±7	55±19	**85±8**	83±4	51±16	**87±7**	80±7	57±14	**77±10**	74±17	39±19
$GConv_{u2}$	**85±5**	70±11	54±10	**81±12**	69±8	52±14	83±4	71±9	61±17	85±11	77±19	59±19	74±19	69±24	37±11
$PointNet++$ [56]	69±17	71±22	80±15	66±19	68±18	76±10	71±17	75±19	74±20	59±22	62±28	68±23	21±12	24±10	20±6
$EVIMO$ [31]		70±5			59±9			78±5			79±6			67±3	

Table 6.1: *Segmentation results on EVIMO event-based dataset. Metric is $mIoU(\%)$ on points; the 3D methods were evaluated on 3 different time-slice widths: 0.3, 0.1 and 0.02 seconds. The best results are shown in bold for every separate validation set type.*

tion set for the first $12k$ iterations of training. The experiments on full resolution clouds (shown in solid lines) perform notably better, as to be expected, but the training speed is 30% lower (for results on training speed see Table 6.2).

We observe that the biggest advantage form increasing the slice width comes for larger slice widths - the $u2, w0.1$ and $u1, w0.1$ perform similarly, while $u1, w0.3$ is 10% better than $u2, w0.3$. Intuitively, this can be due to the lack of temporal features in smaller slices - a similar effect is observed with edge radius (Fig. 6.7-(b)), since the edge radius, together with the network depth, essentially controls the maximum size of global features.

6.4.3.2 Temporal Augmentation

Temporal augmentation is designed to reduce overfitting of the network to the local 3D feature descriptors by randomly scaling the event cloud along time axis. The *EVIMO* dataset has high variation in object velocities, including across training and validation sets; the velocity of the object defines the shape of the cloud (as explained in Sec. 6.2.1), which can cause significant drop in scores when transferring to unseen se-

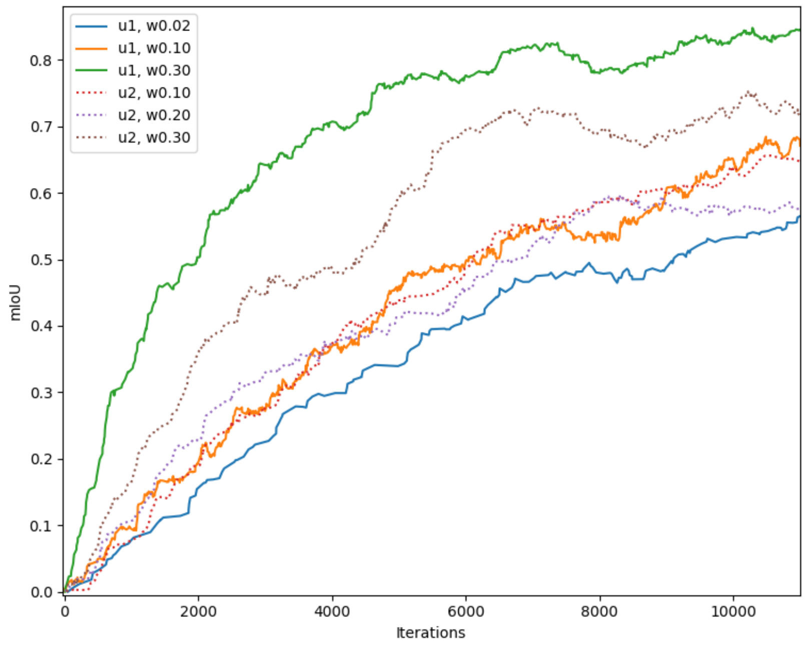
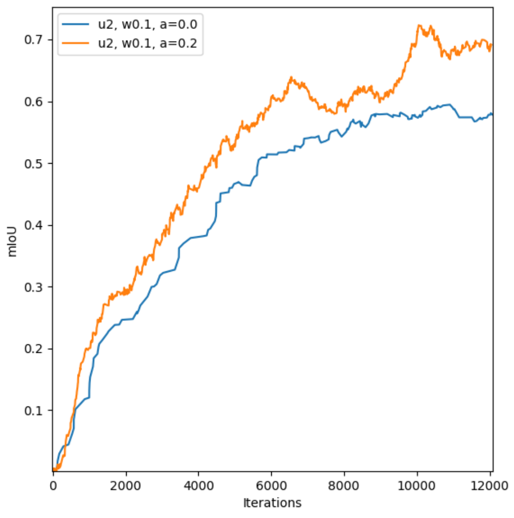


Figure 6.6: *Comparison of inference performance, in $mIoU$, for the first $12k$ iterations of training for slice widths $w = 0.02, 0.1, 0.3$sec. Dashed line corresponds to subsampling factor $u = 2$. The solid line corresponds to experiments with no subsampling ($u = 1$).*

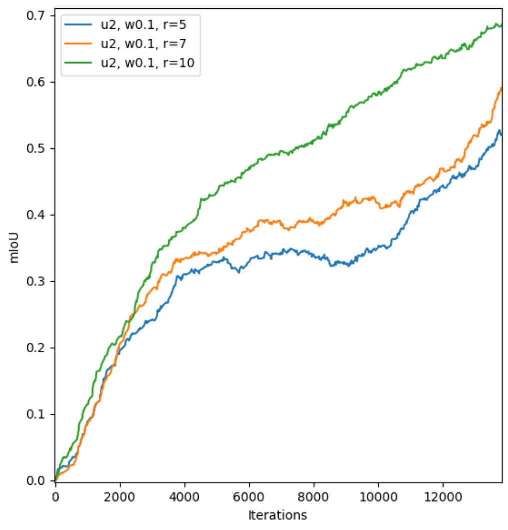
quences in validation set. We show the comparison for learning with and without augmentation in Fig. 6.7-(a) by training on the *boxes* dataset and evaluation on *fast* dataset.

6.4.4 Results

Both $GConv$ and $PointNet++$ were trained on all *EVIMO* train sequences containing objects: *boxes*, *floor*, *wall* and *table* and evaluated separately on *boxes*, *floor*, *wall*, *table* and *fast* validation sets with the 0.02 sec. temporal step between slices, $\alpha = 0.2$ for temporal augmentation (disabled for validation), subsampling rates of $u = 1$ and $u = 2$ and slice widths $w = 0.02$, 0.1 and 0.3 sec. The quantitative results are presented in



(a)



(b)

Figure 6.7: *$mIoU$ scores on the box validation set by disabling temporal augmentation and reducing the maximum edge radius. (a) - The temporal augmentation is described in Sec. 6.3.4. Plots are shown for $\alpha = 0.0$ (no augmentation) and $\alpha = 0.2$ for the first $12k$ iterations of training. (b) - The maximum edge radius is reduced from 10 to 7 and 5, which results in significant performance drop.*

Table 6.4.2.

For EVIMO method, which outputs dense masks, we set the slice width at $0.025sec$. We up-project the inferenced masks on the event cloud (similar to how we handle ground truth, Sec. 6.4.1.1) and compute IoU scores on the labeled event cloud.

6.4.4.1 Qualitative Results

We show qualitative results in Fig. 6.5 by downprojecting events along time axis. The color corresponds to event timestamp (blue is 0, red is slice width). The top section of the figure compares results with no subsampling for a 0.02 slice and 0.3 slice. Note how significantly more prominent motion of the object on larger slice improves segmentation quality even in textured regions. The subsampling factor $u = 2$, which removes half the points from event cloud also produces high quality results, but is more prone to false positives. For the bottom section of the figure, 80% of the points were removed. The quality of input suffers significantly, with many spatial and temporal features lost, and the network performs poorly in high-textured regions.

6.4.4.2 Performance Considerations

We present the forward timings for our GConv and for PointNet++ across different time slice widths and event cloud subsampling factors in Table 6.2, measured on a single NVIDIA GTX1080Ti. In all experiments, PointNet++ is notably slower than GConv, but starting from 0.1 second slice GConve is not real time as well without subsampling. This problem can be practically addressed by constructing a lookup table [61] and avoid recomputing features when sequential time slices overlap.

6.5 Conclusion

Our approach operates on large slices in time - this allows the neural network to observe a history of scene motion, and potentially make better decisions based on the global temporal features - all without relying on LSTM-like approaches (which essentially memorize scene content). On the other hand, modern event-based cameras are capable of generating up to 10^7 events per second, and with increased resolution, the amount

	$GConv$		$PointNet++$		
params	117.5k		1.4M		$\#pts$
u	1	2	2	5	
20ms	0.016	0.012	0.219	0.134	23073
0.1s	0.109	0.048	4.260	0.663	170503
0.3s	0.275	0.140	22.93	6.792	417315

Table 6.2: *Forward time (seconds) for different uniform downsampling and temporal slice widths of event cloud.*

of local edge connections between graph nodes can grow exponentially. In this work, we were able to achieve slice widths of up to 0.3 seconds (and 0.5 seconds with severe downsampling) due to the large amount of memory consumed by 3D points. A dedicated GPU-optimized module could alleviate many of the shortcomings we have witnessed. We also observe that edges are useful mostly for extracting local features, and hence a dual neural network - one to extract local features on thin temporal slices and another to extract temporal features on large slices will be among our future efforts.

Chapter 7: Event-based constraints on camera egomotion, event surface normals and depth

7.1 Introduction

This chapter presents a mathematical formulation and a set of constraints for event-based vision, and to the best of authors' knowledge is the first to analyze the problem of event-based processing without relying on feature detection and tracking or conversion to frame-like representations. The derived constraints for a monocular event camera connect the per-event scaled depth, full optical flow and camera egomotion and allow to use depth positivity or photometric error functions in either minimization or learning frameworks. We also present a constraint on second order derivatives of event surface, which can be directly used to optimize for camera egomotion.

7.1.1 Motion Estimation and Event Cloud Structure

There is a number of works which attempt to recover scene motion from the 3D, (x, y, t), structure of the event cloud. Local methods [13–16] are often focused on normal flow estimation and usually based on event surface normal estimation. Egomotion can be recovered from flow, which has been explored for classical cameras as well, for example in [62]. A number of matching [17] and tracking [18, 22] methods are capable of computing full optical flow and producing tracks which could be used for egomotion estimation. On the other hand, global methods [19, 20] apply a model to the entire pixel space, but can only produce results locally in time.

7.2 Preliminaries

In the context of event-based sensors, every visible moving contour on the scene produces a trail of events on the image plane of the camera, referred to as *time-surface* [1] or *event cloud* [31]. This trail consists of points $\{\mathrm{x}, \mathrm{y}, t\}$, with pixel coordinates x and y, while timestamp t specifies the exact moment when the edge passes through that pixel.

7.2.1 Event Surface Parameterization

Without the loss of generality, let us assume that all contours on the scene can be described in terms of $\hat{X} = \hat{X}(\sigma)$ - a 3D curve given in the camera frame at the initial moment of time $t = 0$, and parameterized by its arc length σ. For a given contour $\hat{X}$, if the motion of the camera $R = R(t)$ and $T = T(t)$ is given, the triplet

$$\{\hat{X}(\sigma), R(t), T(t)\} \tag{7.1}$$

completely describes the Event Surface geometry produced by the given contour (this statement will be further explored below). By combining Event Surfaces from all visible edges the complete event cloud can be described. This parameterization does not describe occluded regions, but this will not be significant in our future derivations; an analysis of contours in presence of occlusions is conducted by Åström in [63].

7.2.2 Notation

We define the *world coordinate frame* as a coordinate frame of the camera at $t = 0$, all variables given in this frame are marked by $\hat{}$ symbol. The lowercase $x = \begin{bmatrix} \text{x} & \text{y} \end{bmatrix}^T$ defines normalized pixel coordinates on the camera plane. We define the time derivative $\frac{d}{dt}$ of a function $f(t)$ with $\dot{f}$; to avoid confusion, for multivariate functions $\dot{f}$ defines a partial derivative, with other variables fixed. Following this, $\dot{x}(\sigma, t) = \frac{d}{dt}x(\sigma, t)|_{\sigma=const}$ defines the derivative of the pixel trajectory as the contour moves relative to the camera, which we refer to as '*optical flow*'.

The parameterization of the camera motion is defined with respect to a particular rigid contour $\hat{X}(\sigma)$, and is given as $\{R(t), T(t)\}$, such as

$$\begin{gathered} X(\sigma, t) = R(\hat{X} - T), \quad \dot{X}(\sigma, t) = \Omega X + V \\ R(0) = I \qquad T(0) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \end{gathered} \tag{7.2}$$

Here X is the 3D coordinate of the contour point σ in the local camera frame at time t, Ω is a skew-symmetric matrix specifying the angular velocity of the camera, and V is a column vector specifying its linear velocity. The derivations for Ω and V are presented in Sec. 7.7. It is worth mentioning, that while $\hat{X}$, R and T are defined in the world coordinate frame, X, Ω and V are in the local camera frame.

7.3 Constraint on Event Surface Normals

The property of the optical flow to be orthogonal to Event Surface normals has been explored in prior works [13–16]. Indeed, vector $\dot{x} = \begin{bmatrix} \dot{\text{x}} & \dot{\text{y}} & \dot{t} \end{bmatrix}^T = \begin{bmatrix} \dot{\text{x}} & \dot{\text{y}} & 1 \end{bmatrix}^T$ (being a derivative of the surface) is tangential to the Event Surface.

Here we derive a constraint which connects scene depth and the shape of the event cloud. By defining $n = \begin{bmatrix} n_x & n_y & n_z \end{bmatrix}$:

$$0 = \begin{bmatrix} \dot{\text{x}} & \dot{\text{y}} & 1 \end{bmatrix}^T \cdot \begin{bmatrix} n_x & n_y & n_z \end{bmatrix} \implies \begin{bmatrix} -\frac{n_x}{n_z} & -\frac{n_y}{n_z} \end{bmatrix} \dot{x} = 1 \tag{7.3}$$

Let $c(\sigma, t) = \begin{bmatrix} -\frac{n_x}{n_z} & -\frac{n_y}{n_z} \end{bmatrix}$. Using Eq. 7.7 from Sec. 7.4 and Eq. 7.3:

$$\begin{gathered} c\dot{x} = 1, \qquad \dot{x} = A(\Omega b + \frac{V}{\text{Z}}) \implies cA(\Omega b + \frac{V}{\text{Z}}) = 1 \\ \implies \text{Z} = \frac{cAV}{1 - cA\Omega b} \end{gathered} \tag{7.4}$$

Eq. 7.4 is fundamental in a sense that it connects local normal of the Event Surface and depth; in other words, the shape of the Event Surface in (x, y, t) space can be connected to the geometry of the scene through the camera instantaneous velocity.

7.4 Optical Flow Equation

It is important to notice, that our definition of *optical flow*, given in Sec. 7.2.2, does not completely coincide with its classical definition as a vector field warping image pixels between to discrete moments in time t_0 and t_1. The connection between the two definitions (where $\tilde{f}$ is the discrete flow) can be expressed as:

$$\tilde{f} = \int_{t=t_0}^{t=t_1} \dot{x}(\sigma, t) dt \tag{7.5}$$

Let $X = \begin{bmatrix} \mathrm{X} & \mathrm{Y} & \mathrm{Z} \end{bmatrix}^T$; define projection function $\mathcal{P}(X)$

$$\begin{gathered} \mathcal{P}(X) = \begin{bmatrix} \frac{\mathrm{X}}{\mathrm{Z}} & \frac{\mathrm{Y}}{\mathrm{Z}} \end{bmatrix}^T \qquad \text{then} \qquad x = P(X) \\ \dot{x} = \frac{dP}{dX}\dot{X} = \frac{1}{\mathrm{Z}} \begin{bmatrix} 1 & 0 & -\mathrm{x} \\ 0 & 1 & -\mathrm{y} \end{bmatrix} (\Omega X + V) \end{gathered} \tag{7.6}$$

Or, by defining

$$\begin{gathered} A = \begin{bmatrix} 1 & 0 & -\mathrm{x} \\ 0 & 1 & -\mathrm{y} \end{bmatrix}, \qquad b = \begin{bmatrix} \mathrm{x} & \mathrm{y} & 1 \end{bmatrix}^T \\ \dot{x} = A(\Omega b + \frac{V}{\mathrm{Z}}) \end{gathered} \tag{7.7}$$

We can now substitute the results acquired in Eq. 7.4:

$$\begin{gathered} \dot{x} = A\Omega b + \frac{1}{\mathrm{Z}} AV, \qquad \mathrm{Z} = \frac{cAV}{1 - cA\Omega b} \\ \dot{x} = A\Omega b + (1 - cA\Omega b)\frac{AV}{cAV} \end{gathered} \tag{7.8}$$

The Eq. 7.8 is the *optical flow equation for Event Surfaces*, and does not depend on the scene geometry.

7.5 Constraint on the Second Order Derivatives

In this section we investigate the second order derivative of the pixel trajectory.

7.5.1 Derivative of Optical Flow

Following Eq. 7.6 and Eq. 7.2:

$$\frac{d^2x}{dt^2} = \frac{d}{dt}\frac{dP}{dX}\dot{X} + \frac{dP}{dX}\ddot{X} \tag{7.9}$$

Here and below all temporal derivatives are partial. Evaluating the first term of Eq. 7.9:

$$\begin{aligned}
\frac{d}{dt}\frac{dP}{dX}\dot{X} = \frac{d}{dt}(\frac{1}{Z}A)\dot{X} &= \begin{bmatrix} -\frac{\dot{Z}}{Z^2} & 0 & -\frac{\dot{X}}{Z^2} + \frac{2X\dot{Z}}{Z^3} \\ 0 & -\frac{\dot{Z}}{Z^2} & -\frac{\dot{Y}}{Z^2} + \frac{2Y\dot{Z}}{Z^3} \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \\
= \begin{bmatrix} -\frac{\dot{Z}\dot{X}}{Z^2} - \frac{\dot{Z}\dot{X}}{Z^2} + \frac{2X\dot{Z}^2}{Z^3} \\ -\frac{\dot{Z}\dot{Y}}{Z^2} - \frac{\dot{Z}\dot{Y}}{Z^2} + \frac{2Y\dot{Z}^2}{Z^3} \end{bmatrix} &= -\frac{2\dot{Z}}{Z^2} \begin{bmatrix} \dot{X} - x\dot{Z} \\ \dot{Y} - y\dot{Z} \end{bmatrix} = \\
= -\frac{2\dot{Z}}{Z^2}A\dot{X} = -\frac{2\dot{Z}}{Z}\frac{dP}{dX}\dot{X} = -\frac{2\dot{Z}}{Z}\dot{x}
\end{aligned} \tag{7.10}$$

Using the definition of $\Omega = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix}^T_\times$:

$$\begin{aligned}
\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} &= \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} \\
\implies \frac{\dot{Z}}{Z} &= w_1 y - w_2 x + \frac{V_z}{Z}
\end{aligned} \tag{7.11}$$

Returning to Eq. 7.9:

$$\begin{aligned}
\ddot{X} &= \dot{\Omega}X + \Omega\dot{X} + \dot{V} = \dot{\Omega}X + \Omega(\Omega X + V) + \dot{V} \\
\ddot{x} &= -\frac{2\dot{Z}}{Z}\dot{x} + A\dot{\Omega}b + A\Omega(\Omega b + \frac{V}{Z}) + A\frac{\dot{V}}{Z}
\end{aligned} \tag{7.12}$$

7.5.2 Derivative of Surface Normal Constraint

Now we will analyze the constraint on the Event Surface normals. From Eq. 7.3:

$$1 = c \cdot \dot{x} \implies \dot{c}\dot{x} + c\ddot{x} = 0$$

$$\begin{bmatrix} \dot{c_1} & \dot{c_2} \end{bmatrix} = \begin{bmatrix} \dot{x} & \dot{y} \end{bmatrix} \begin{bmatrix} \frac{d}{dx}c_1 & \frac{d}{dx}c_2 \\ \frac{d}{dy}c_1 & \frac{d}{dy}c_2 \end{bmatrix} = \dot{x}^T D \tag{7.13}$$

$$\implies \dot{x}^T D \dot{x} + c\ddot{x} = 0$$

7.5.3 Second Order Constraint on Event Surfaces

Finally, by combining Eq. 7.10- 7.13, Eq. 7.8 and Eq. 7.4:

$$0 = \dot{x}^T D\dot{x} - 2(w_1 y - w_2 x + \frac{V_z}{Z}) + cA(\dot{\Omega} + \Omega^2)b + cA\Omega\frac{V}{Z} + cA\frac{\dot{V}}{Z} \tag{7.14}$$

or, using Eq. 7.29:

$$0 = \dot{x}^T D\dot{x} - 2(w_1 y - w_2 x) + 2\frac{cA\Omega V - V_z}{Z} + cA\ddot{R}R^{-1}b - cAR\frac{\ddot{T}}{Z} \tag{7.15}$$

7.6 Experiments

7.6.1 Dataset

We have collected a dataset using the latest 640×480 camera from Prophesee, which in our setup generates $7M$ events per second. The dataset was collected with a handheld camera indoors and features a variety of motions across all 6 degrees of freedom. The ground truth is provided by Vicon motion capture system, which has been calibrated with the camera center. We use a lens with 100° horizontal field of view. We show several frames from hi-res dataset on Fig. 7.2.

The equations 7.4, 7.8 and 7.14 require surface derivatives (c and D) and event coordinates (A and b) to be determined with high accuracy - in particular due to our requirement in derivation of 7.4, that optical flow is orthogonal to surface normals. To improve the quality of event surfaces we apply the radius outlier filter with threshold of 20 points, within an ellipsoid or 3 pixels by 7 milliseconds in (x, y, t) space. We further reconstruct the surface using a *Moving Least Squares* (MLS) method, by fitting locally cubic polynomials - we show the comparison for the data before and after the preprocessing stage on Fig. 7.3. Parameters c and D can be directly determined from the coefficients of the fitted polynomials.

7.6.2 Error Functions

We design a set of error functions based on the Eq. 7.4, 7.8 and 7.14. All of these functions are *instantaneous in time*, which means that they are valid for an infinitely small time slice of events, and instantaneous linear and angular velocities V and Ω. In practice, a large time slice can be used, in conjunction with spline-interpolated set of instantaneous velocities in a global optimization framework.

7.6.2.1 Depth Positivity Error

The simplest error function is based on Eq. 7.4, and is just a percentage of points with non-positive depth values in the slice, given an egomotion input:

$$E_{dp}(\Omega, V) = \frac{1}{N}\sum \delta(x_i, \Omega, V), \qquad \delta(x_i, \Omega, V) = \begin{cases} 0 & Z(x_i, \Omega, V) < 0 \\ 1 & Z(x_i, \Omega, V) \geq 0 \end{cases} \tag{7.16}$$

We show the plot of this function on Fig. 7.4 - (a); this constraint is the weakest and is unstable for small values of linear velocity V. Provided that the number of events is large enough, the function is smooth and has a gradient.

7.6.2.2 Photometric Error

The analogue of a classical photometric error in (x, y, t) space would measure the similarity between two event clouds corrected with optical flow at two distinct moments of time. Here we use Eq. 7.8 to compute optical flow $f(\Omega, V)$; we motion-compensate the event cloud with optical flow (similar to [19]) and voxelize the event cloud:

$$I_j = \frac{1}{N_j}\sum (x_i + f_i), \qquad where \qquad x_i \in \hat{I}_j \tag{7.17}$$

In practice, we choose voxel regions $\hat{I}_i$ to be the size of 1 pixel and with temporal height of the event slice - this approach results in two event maps I^- and I^+, the first one built with events below the average slice timestamp, and the other one is with events above. We compute the mean distance between corresponding cells of I^- and I^+:

$$E_{photo} = \frac{1}{N}\sum |I_j^- - I_j^+|_2 \tag{7.18}$$

The plot of this function is given in Fig. 7.4 - (b). It is worth mentioning that this error is a measure of the similarity of pixel motion within a small time interval, and is not related to the intensity or contrast of the image, the name *photometric* here is for simplicity - since the idea of comparing the pixel intensities over two frames for conventional cameras is similar to the idea of comparing pixel motion for event-based cameras.

7.6.2.3 Variance Error

We design *Variance Error* in the same spirit as the photometric error, but in addition we use the notion that the variance of events along the direction orthogonal to the edge (along local surface normal) should be minimal given the correct optical flow estimate. Similar to Eq. 7.17 we construct I (a single map for the entire slice) and measure the variance of the event distribution within each cell $\hat{I}_i$ along the mean normal direction $\bar{n}$:

$$\begin{gathered} I_j^{var} = \frac{1}{N_j} \sum |(x_i + f_i - I_j) \cdot \bar{n}|^2, \qquad where \qquad x_i \in \hat{I}_j \\ E_{var} = \frac{1}{N} \sum I_j^{var} \end{gathered} \tag{7.19}$$

Although slightly more difficult to compute, our experiments show that the variance error provides a sharper gradient than the photometric error. We plot variance error as a function of linear velocity on Fig. 7.4 - (c).

7.6.2.4 Second Order Error

All previous error functions are approximations - they require an event slice large enough to apply optical flow for correction, and thus require the assumption that both the camera velocity and optical flow remain approximately constant within a small time interval. Moreover, optical flow in Eq. 7.8 is instantaneous and is tangential to event surface, and hence this flow does not provide the true pixel-to-pixel mapping if surface has non-zero curvature.

The only *truly instantaneous* constraint is Eq. 7.14, which provides a solid criteria for egomotion correctness and does not require extrapolations.

$$\begin{aligned} g(\Omega, V, \dot{\Omega}, \dot{V}) = \dot{x}^T D \dot{x} - 2(w_1 y - w_2 x + \frac{V_z}{Z}) + cA(\dot{\Omega} + \Omega^2)b + cA\Omega\frac{V}{Z} + cA\frac{\dot{V}}{Z} \\ E_{so} = \frac{1}{N}\sum g_i(\Omega, V, \dot{\Omega}, \dot{V})^2 \qquad \textit{for all} \quad x_i \quad \textit{in event cloud} \end{aligned} \tag{7.20}$$

We plot this function on Fig. 7.4 - (d); the downside of this approach is twofold: first, it is a second order constraint and required matrix D - the second order derivative of the event surface, which is difficult to compute and which is not robust against noise. Another downside is the double number of parameters in comparison to flow-based or depth-based errors. As a result, this method is only feasible as a refinement step after approximate egomotion is determined. The parameter count can also be reduced via spline-based trajectory estimation, where all derivatives of the trajectory are easy to compute.

7.6.3 Results

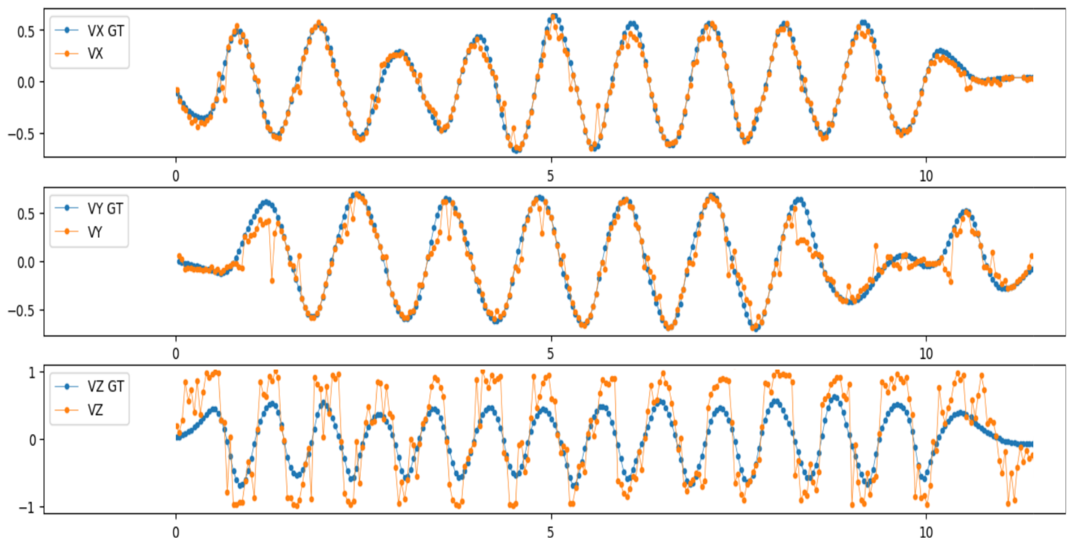


Figure 7.1: *Motion estimation with a self-supervised learning on 3D pointcloud, using the variance error, Eq. 7.19, as loss. The camera resolution is* 640×480 *and is handheld; the ground truth (in blue) is collected using Vicon motion capture system. Note the loss of accuracy on the Z axis - a consequence of flow equation (with depth substituted from Eq. 7.4) having very small gradient along Z axis.*

To demonstrate the practical applications of the new error functions we use our ego-motion dataset, collected with a handheld Prophesee 640×480 camera and featuring fast 6 DoF motion and high accelerations. The learning pipeline was adapted from Chapter 6, with loss function replaced by Variance Error (Eq. 7.19); the Photometric Error (Eq. 7.18) achieves similar results in our experiments. This learning is completely *self-suprevized* and requires no ground truth, We show the inference results on Fig. 7.1.

7.7 Camera Motion Model

In our implementation, the ground truth from motion capture system, as well as all estimates for system rotation are represented as Euler angles, in $x-y-z$ format. The Euler angle representation is chosen as it provides for a more intuitive way to filter the data using univariate splines for each axis of rotation. Specifically the rotation matrix R is defined as:

$$\begin{aligned} R &= R_x \cdot R_y \cdot R_z = \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix} \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{7.21}$$

Lets define $\hat{X}$ as a 3D coordinate of the scene point in camera coordinate frame at $t=0$, when camera rotation $R=I$ and translation $T = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$; we will define with $X = X(t)$, $\hat{X} = X(0)$ the coordinate of the same point in the camera coordinate frame at time t. Furthermore, let R and T be defined such that:

$$X = R(\hat{X} - T) \tag{7.22}$$

Then

$$\begin{aligned} \dot{X} &= \dot{R}(\hat{X} - T) - R\dot{T} \\ \dot{X} &= \dot{R}R^{-1}X + (-R\dot{T}) \end{aligned} \tag{7.23}$$

Define $\Omega = \dot{R}R^{-1}$, and $V = -R\dot{T}$, then:

$$\dot{X} = \Omega X + V \tag{7.24}$$

7.7.1 Derivations for Angular Velocity and Acceleration

Let $\alpha = \alpha(t)$, $\beta = \beta(t)$ and $\gamma = \gamma(t)$ be the Euler angles of camera rotation around X, Y and Z axes (see Eq. 7.21) in the initial coordinate frame of the camera at $t = 0$. Evaluating $\dot{R}_x$ at $t = t_0$:

$$\dot{R}_x = \dot{\alpha} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & -sin(\alpha) & -cos(\alpha) \\ 0 & cos(\alpha) & -sin(\alpha) \end{bmatrix} = \dot{\alpha} \cdot R_x \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{7.25}$$

Here for brevity $\alpha = \alpha(t_0)$ and $\dot{\alpha} = \dot{\alpha}(t_0)$. Define:

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \mathcal{B} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \mathcal{C} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{7.26}$$

Then, following the derivation of Eq. 7.25, it is easy to show that:

$$\begin{array}{lcl} \dot{R}_x = \dot{\alpha} R_x \mathcal{A} & & R_x \mathcal{A} = \mathcal{A} R_x \\ \dot{R}_y = \dot{\beta} R_y \mathcal{B} & and & R_y \mathcal{B} = \mathcal{B} R_y \\ \dot{R}_z = \dot{\gamma} R_z \mathcal{C} & & R_z \mathcal{C} = \mathcal{C} R_z \end{array} \tag{7.27}$$

Finally, from Eq. 7.21 and Eq. 7.23:

$$\Omega = \dot{R}R^{-1} = (\dot{R}_x R_y R_z + R_x \dot{R}_y R_z + R_x R_y \dot{R}_z) \cdot R^{-1} = \\ = (\dot{\alpha}\mathcal{A}R + \dot{\beta}R_x\mathcal{B}R_y R_z + \dot{\gamma}R\mathcal{C}) \cdot R^{-1} = \dot{\alpha}\mathcal{A} + \dot{\beta}R_x\mathcal{B}R_x^{-1} + \dot{\gamma}R\mathcal{C}R^{-1} \tag{7.28}$$

The derivatives $\dot{\Omega}$ and $\dot{V}$ can be expressed as:

$$\begin{aligned} &V = -R\dot{T} \qquad \Omega = \dot{R}R^{-1} \\ &\dot{V} = -\dot{R}\dot{T} - R\ddot{T} = -\Omega R\dot{T} - R\ddot{T} = \Omega V - R\ddot{T} \\ &\dot{\Omega} = \ddot{R}R^{-1} - \dot{R}R^{-1}\dot{R}R^{-1} = \ddot{R}R^{-1} - \Omega^2 \end{aligned} \tag{7.29}$$

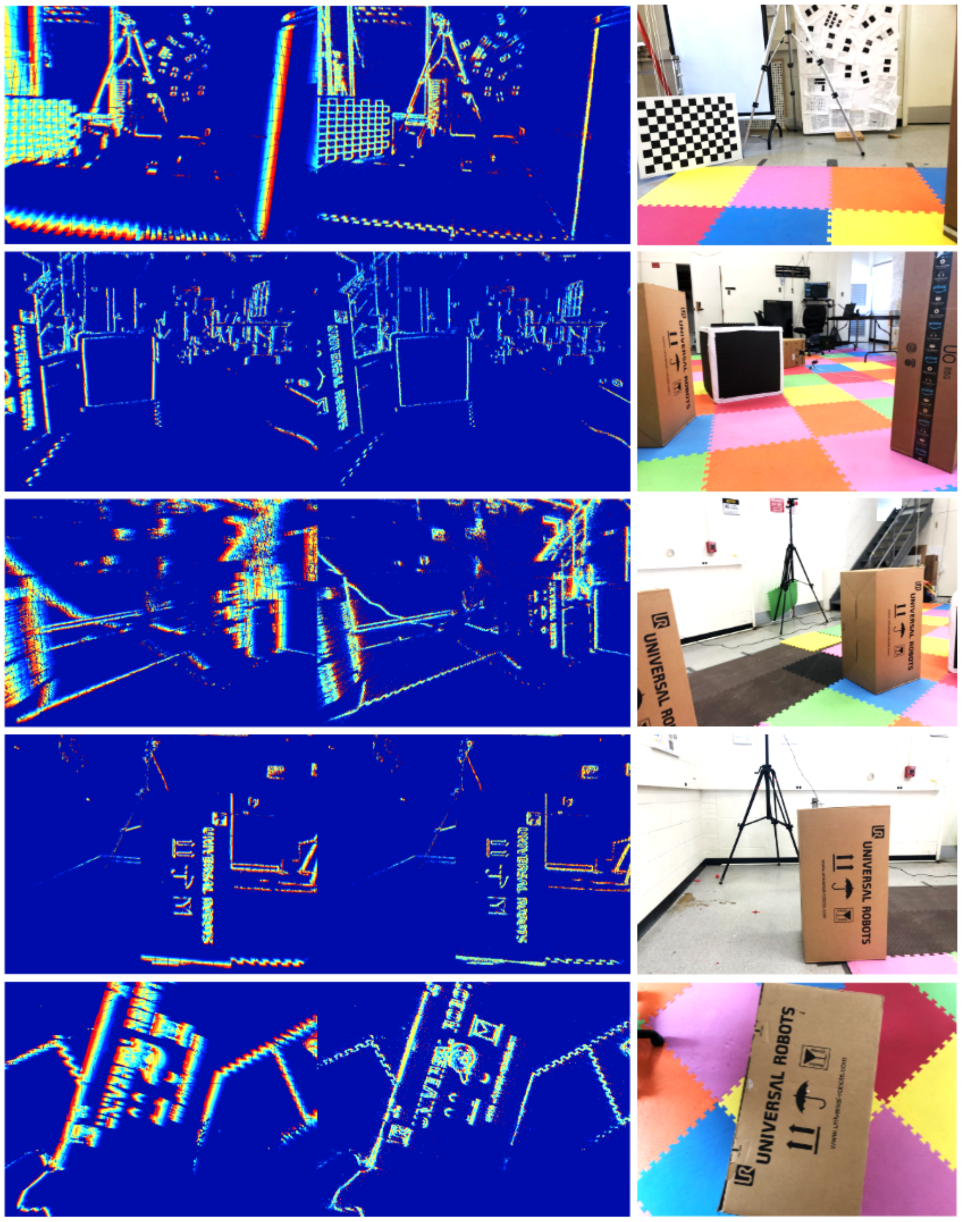


Figure 7.2: *Dataset for egomotion estimation collected with a* 640 × 480 *Prophesee Gen3 VGA event camera. Left - input time slice, 0.04 sec. wide (color gradient shows event timestamp); middle - warped time slice, motion-corrected using optical flow (Eq. 7.8) after optimization for egomotion. Right - classical RGB frame.*

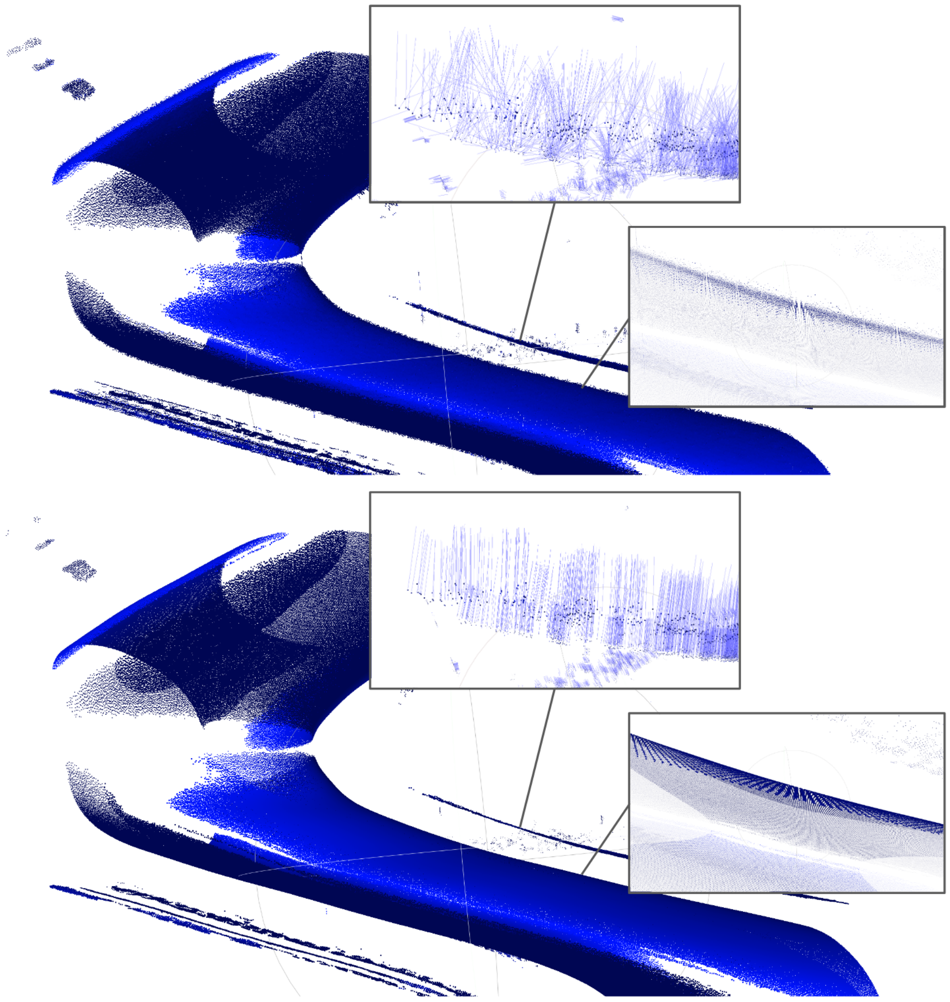
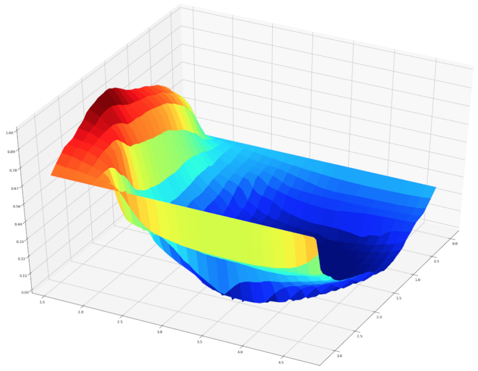
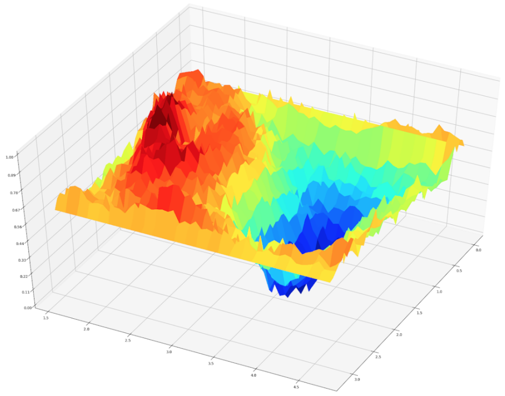


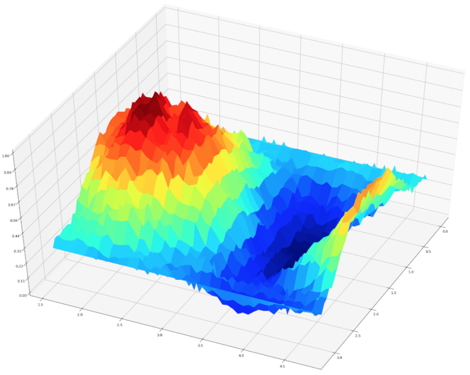
Figure 7.3: *The dataset is preprocessed by applying a radius outlier filter to events, with subsequent moving least squares surface reconstruction. Top: raw event data before preprocessing; the normals and points are superimposed for easier comparison. Bottom: event cloud after preprocessing stage. Note the increase of event density on the surface and reduction of the noise in normals.*



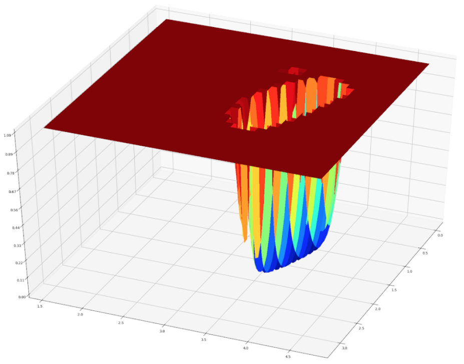
(a) depth positivity error



(b) photometric error



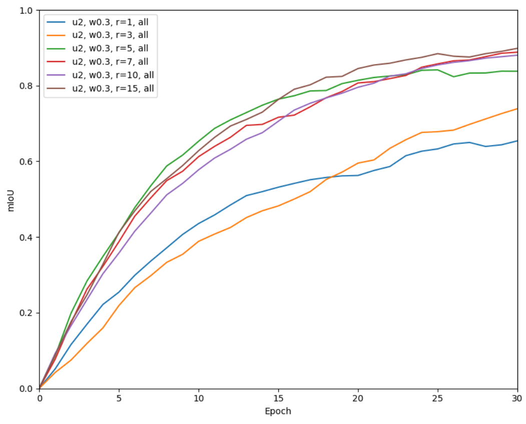
(c) variance error



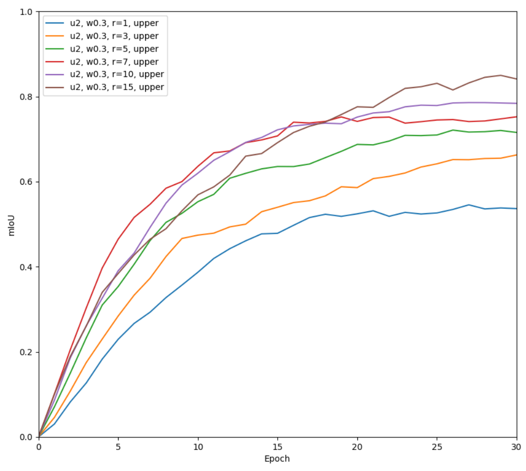
(d) second order constraint

Figure 7.4: *Error function behavior parameterized by the direction of the linear velocity vector in spherical coordinates. The rotation is set to ground truth; in addition, for the second order constraint accelerations are set to ground truth values.*

Appendix A: Learning Visual Motion Segmentation using Event Surfaces - Additional Experiments



(a)



(b)

Figure A.1: *The effects of using edges computed within different radii* r. *(a) - all edges within a sphere are used; (b) - only edges in the upper hemisphere along time axis are used. We show the plots for the first 30 epochs on 'boxes' validation set, with subsampling* $u = 2$ *and slice width* $w = 0.3$ *sec.*

A.1 Edge Configurations - Ablation Study

Our main results were computed with 'upper hemisphere' edge configuration; the motivation to use only edges in the upper hemisphere is the ability to remove half the edges around each point, significantly reducing memory footprint on GPU. We perform additional ablation studies to compare how full (all edges in the sphere) and upper hemisphere edge configurations affect the quality of training for different values of r. All experiments use *boxes* validation set, and are trained on *boxes* train set, with $u = 2, w = 0.3$.

On the Fig. A.1 (a) we show $mIoU$ scores for the first 30 epochs of training with the full edge configuration. Starting from $r = 5$ the results do not show significant improvements as r increases. In this setting, each point on the first layer aggregates features in a sphere with diameter $d = 2r$ - from 10 to 30 pixels for $r > 5$, and the network might reach saturation in the amount of locally available information. On the Fig. A.1 (b) we perform similar experiments with our baseline upper hemisphere configuration. The results improve as r increases, but IoU is slightly lower than for the full configuration.

The experiments with $r = 1$ show the performance of the network with no spatial connections - since pixel coordinates are discretized, only connections along temporal axis can exist. We show additional side-by-side edge configuration comparisons for each radii on Fig. A.3; the per-step runtime for these experiments is shown in Table A.1.

A.2 Training Time

We show additional timing measurements in Table A.1, with upper hemisphere and full configurations, for $u = 2, w = 0.3, r = 1..15$. Although slowdown is expected with full configuration, the main reason behind using only the upper hemisphere was high memory consumption with small u values and large r.

r	1	3	5	7	10	15
upper	0.293	0.415	0.636	0.748	0.975	1.340
full	0.267	0.669	0.858	0.984	1.282	1.751

Table A.1: *Average time per training step (forward and backward pass), in seconds, with batch size $b = 3$, subsampling $u = 2$, slice width $w = 0.3$ sec. for edge radii $r = 1$ to 15 pixels. $full$ corresponds to all edges in a sphere of radius r, $upper$ corresponds to edges only in the upper hemisphere along time axis. The results were collected using 3 Nvidia GTX 1080Ti GPUs.*

A.3 Additional Qualitative Results

A.3.1 Dataset

A sample of EVIMO dataset (*boxes* validation, sequence 0) is shown on Fig. A.2. The points are colored according the corresponding event polarity in green channel, and object id in red channel. The background is shown in green and blue, and the object motion is visible on the upper part of the image, in white and purple. Note how the shape of the cloud follows scene motion.

A.3.2 Inference with Large *w*

We train a baseline neural network with $w = 0.3, r = 10, u = 2$ and apply it to the cloud with $w = 1.0$. The qualitative result is shown on Fig. A.4 - the ground truth is on the left, and inference is on the right; the object is shown in blue color. We achieve similar $mIoU$ scores on larger slices as on the original ones.

A.3.3 Additional Qualitative Results

Fig. A.5 shows 3D renderings on the segmentation results, on *boxes* validation set. For each figure, inference is on the left, ground truth is on the right; background shown in black. Most of the loss in IoU is due to lower recall, while precision is reasonably high. We attach the *.ply* models for these results together with this supplementary material.

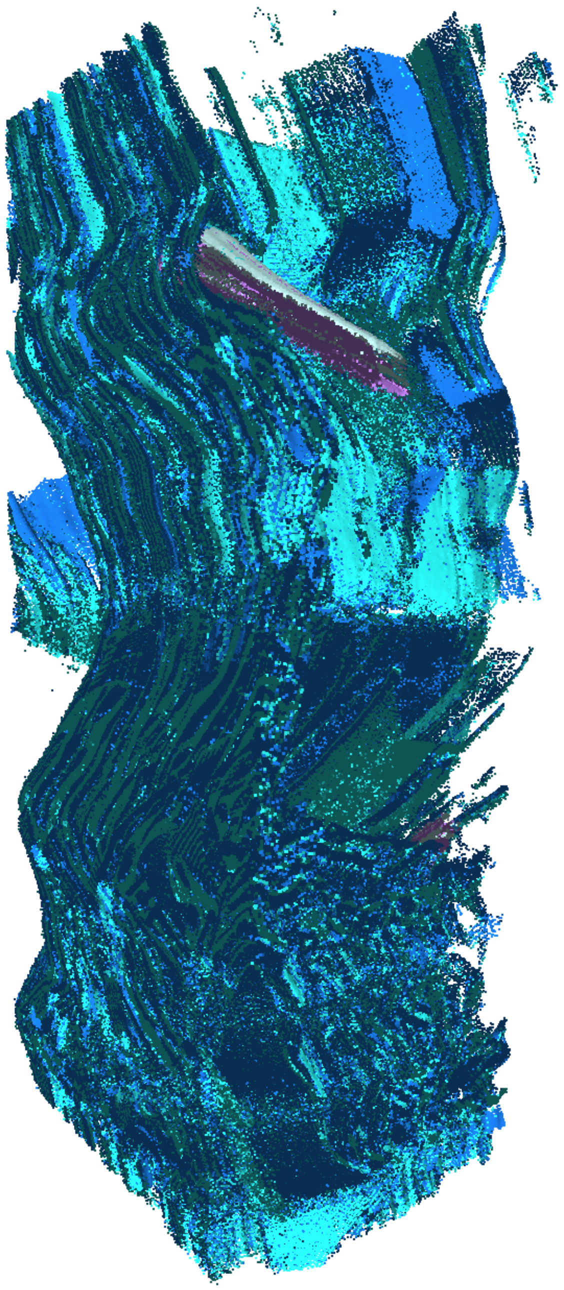
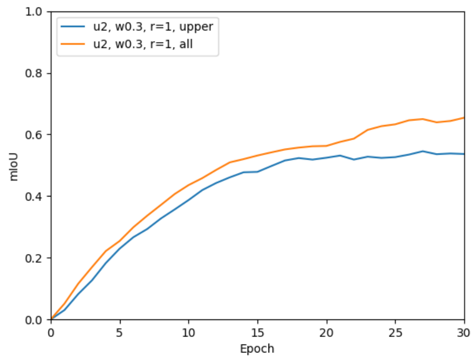
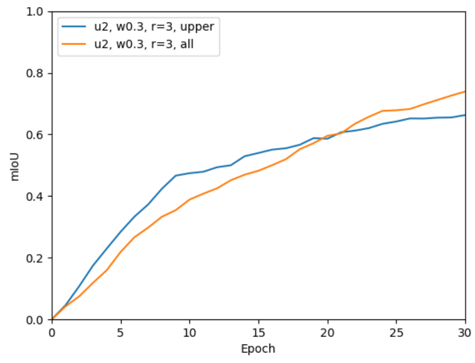


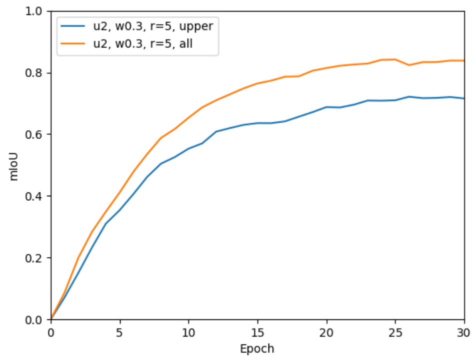
Figure A.2: *A sample of the preprocessed EVIMO dataset, plotted in 3D (time axis is vertical). Red color channel encodes object id, and green encodes event polarity; a separately moving object is visible on the top part of the image in white and purple colors. The green and blue colors represent background.*



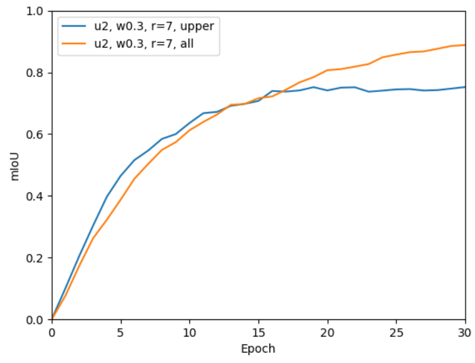
(a)



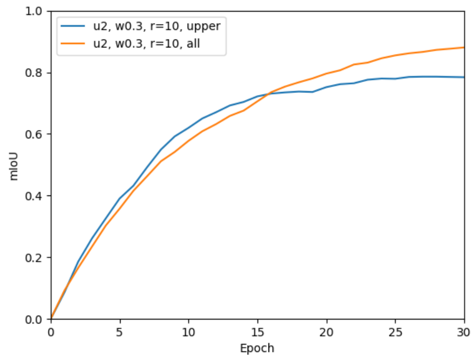
(b)



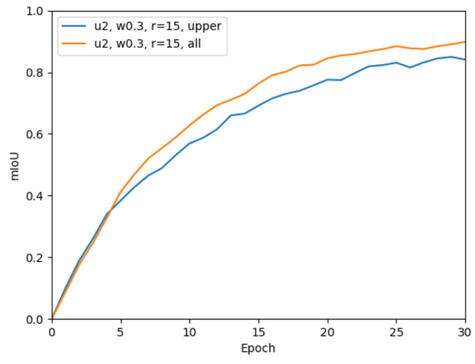
(c)



(d)



(e)



(f)

Figure A.3: *Additional ablation study for full and upper hemisphere edge configurations and for different radii. For $r = 1$ pixels, most edges connect to events only along time axis. We show the plots for the first 30 epochs on 'boxes' validation set, with subsampling $u = 2$ and slice width $w = 0.3$ sec.*

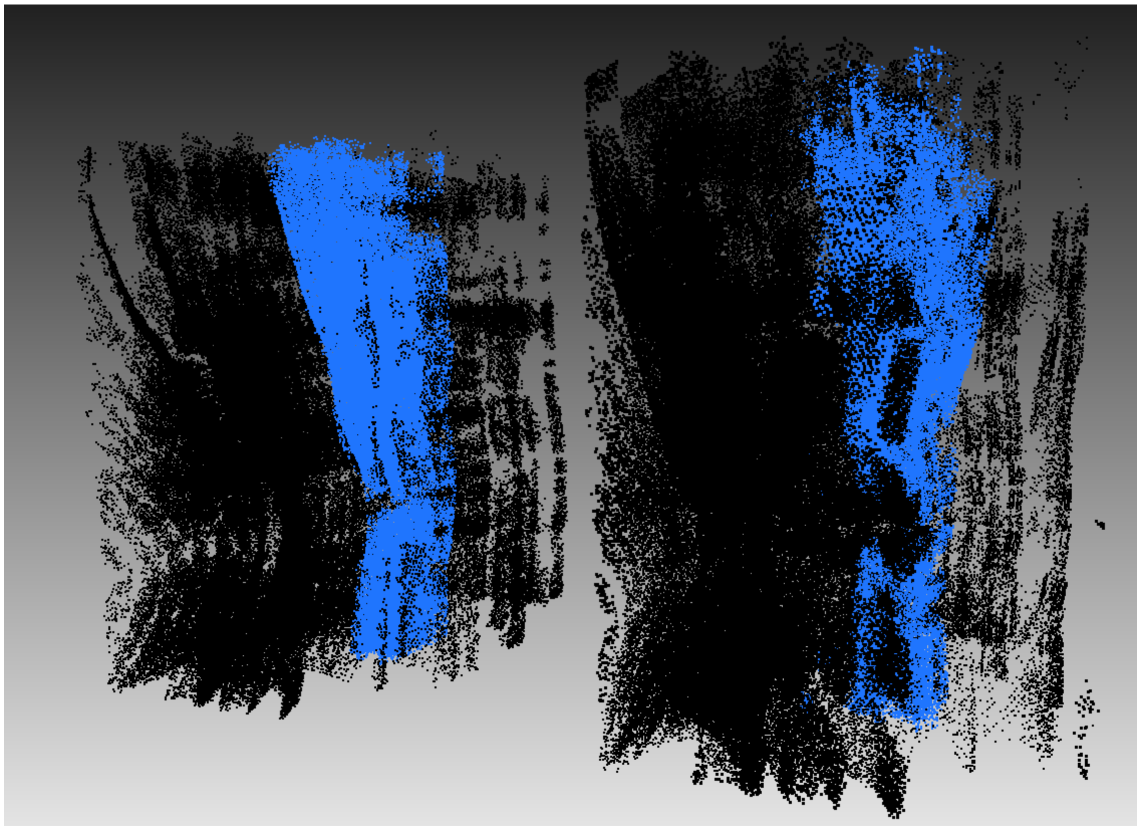
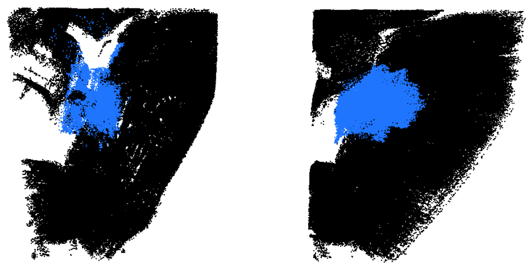
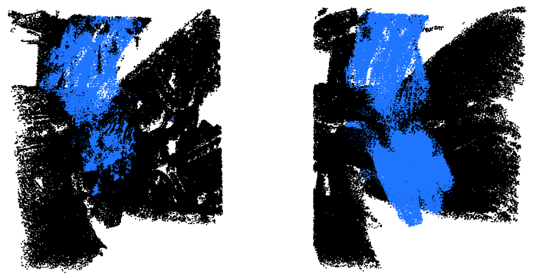
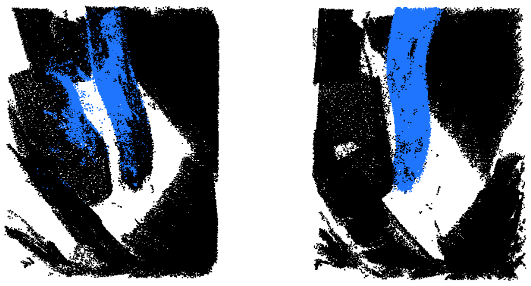
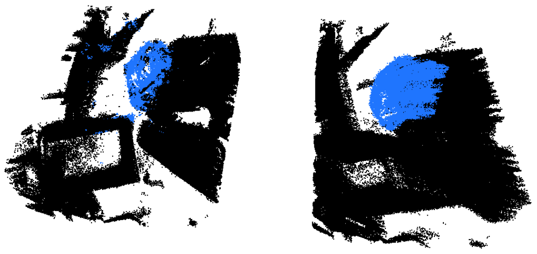


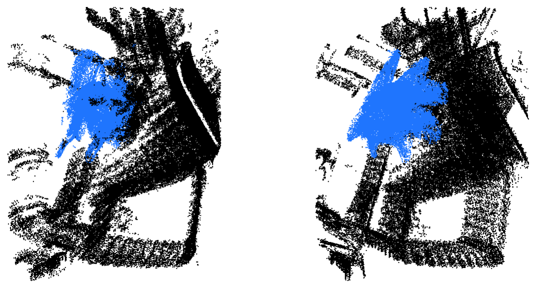
Figure A.4: *Additional qualitative results: the network was trained on* 0.3 *sec. slices with* $r = 10, u = 2$ *and tested on* 1.0 *sec. slices to validate the invariance to slice width. Time axis is vertical; the object is in blue, black color is background. The left image represents ground truth, right is inference.*


(a)


(b)


(c)


(d)


(e)

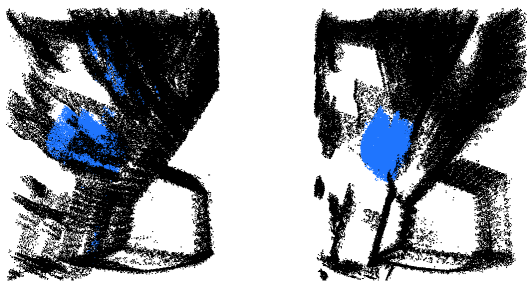

(f)

Figure A.5: *Additional qualitative results: the network was trained with* $w = 0.3, r = 10, u = 2$. *For each figure: left is inference and right is ground truth (two figures per line are shown). Blue represents an object, black is background.*

Appendix B: Towards Hyperdimensional Active Perception

This appendix is based on the paper: A. Mitrokhin, P. Sutor, C. Fermüller, Y. Aloimonos. "Learning Sensori-Motor Control with Neuromorphic Sensors: Towards Hyperdimensional Active Perception", Science Robotics, 2019 [64]

B.1 Introduction

The culprit of modern robotics is the ability to directly fuse the platform's perception with its motoric ability - the concept often referred to as "active perception". Nevertheless, we find action and perception are often kept in separated spaces, which is a consequence of traditional vision being frame-based and only existing 'in the moment', while motion being a continuous entity. This bridge is crossed by the Dynamic Vision Sensor (DVS) - a neuromorphic camera which can see the motion. We propose a method of encoding actions and perceptions together into a single space that is meaningful, semantically informed, and consistent by use of Hyperdimensional Binary Vectors (HBVs). We use DVS for visual perception and show that the visual component can be bound with the systems velocity to enable dynamic world perception, which creates an opportunity for real-time navigation and obstacle avoidance. Actions performed by an agent are directly bound to the perceptions experienced, to form its own "memory".

Furthermore, since HBVs can encode entire histories of actions and perceptions - from atomic to arbitrary sequences - as constant-sized vectors, this is also the first instance of auto-associative memory being combined with deep learning paradigms for controls. We demonstrate these properties on a quadcopter drone ego-motion inference task and the MVSEC dataset.

In recent years, the limits of modern AI and learning have been thoroughly explored in a plethora of datasets, learning architectures, general tasks to solve, and modalities of data. More and more, focus is shifting on what AI can do for modern robotics, and what new forms of it are necessary. What is the science of putting together the different cognitive modalities of an intelligent system, such as vision, motoric actions, audio, and other sensors. Currently, there are no real theories on how to feasibly achieve this, mainly because the representations and processing or learning techniques involved in the different disciplines also differ greatly from each other. Each time a new task is explored, scientists tend to start from the individual components.

One of the more important facets of modern robotics is the notion of integrating the sensory perceptions experienced by an agent with its motoric capabilities - the actions it can perform - to facilitating Active Perception [1, 2], which is considered vital to the existence of autonomous, learning agents that experience the world and are required to interact with it to the best of their abilities [3, 4, 5]. Indeed, the issue of separation of modalities is present in this discipline; the perceptual space and the action space are mainly kept separate, with a central learning mechanism to infer the action given the perception or vice versa. Likewise, the various forms of perception themselves are largely learned or processed separately, embedded into vector representations and essentially concatenated together as input to the aforementioned central learning mechanism. It is clear we must do better than this. Ideally, the perceptions and actions experienced in the moment and in the past should influence the future actions of the agent, and perhaps even bias its expectations for future perception. The main concern is how can we integrate the two vastly different modalities of action and perception together in an effective way.

We propose starting with an integration constraint from the raw data itself; that is, from the beginning we require that perception and action need to eventually "bind" together. For this to happen, there should be some currency through which the perceptual module will interact with the control module or the planning module and so on. This currency is the hyper-dimensional vector, a vector that exists in an extremely high dimensional space. In this paper, we mainly concern ourselves with Hyperdimensional Binary Vectors (HBVs) and the notion of facilitating Hyperdimensional Active Perception (HAP). Whether you are an image, video, motion sequence, control sequence, a concept, word or sound, you are represented by an associated HBV. Furthermore, if you are a sequence of any of these modalities, you are also a HBV of equal dimension constructed by constituent elements of the sequence, thereby existing in the same space as an encoding. Finally, when considered together, such as a mix of modalities or sequences of them, this is also represented by an HBV constructed by binding each modality together, with, yet again, equal dimension. The integration problem now acquires new meaning - all information is represented as long binary vectors that are meaningfully constructed.

While the philosophy of how to handle different modalities of data is important to facilitate active perception, we must also consider the "hardware" aspects. Some sensors are more suited to perceiving information we consider suitable to the problem. Classical vision tends to focus on cameras that are RGB light intensity based, though we find that the biological side tends to interpret signals in very particular ways. One example of this is motion. Motion is not well represented in classical cameras and vision techniques - a consequence of traditional vision being frame-based and only existing ’in the moment’ while motion is a continuous entity. So-called neuromorphic cameras attempt to capture this notion of seeing motion. The more recent development of the Dynamic Vision Sensor (DVS) follows these lines in an interesting way, seeing sparse events in time, as opposed to pixels and intensities. With the introduction of such neuromorphic hardware, we are ready to cross the bridge of frame based vision and develop a new concept - motion-based vision. The event-based sensor provides dense temporal information about scene changes, allowing for accurate, fast, and sparse perception of the dynamic aspect of the world. When it comes to the marriage of action and perception, the fast, asynchronous events that the DVS sees are desirable for facilitating action. As of now, there are no real standards for how to effectively learn from DVS in the manner done for regular RGB cameras - it is all experimental.

Following recent accounts in the field [6] an “active perceiver knows why it wishes to sense, and then chooses what to perceive, and determines how, when and where to achieve that perception”. The “what” question has to do with scene selection and fixation. The “when” question has to do with temporal selection, an instant in time and an extent (a history or episodic memory). The “how” question has to do with mechanical alignment, sensor alignment and priming, and the “where” question has to do with viewpoint selection and the relationship between agent pose, sensor pose and object pose.

All these questions can be addressed using our hyper dimensional framework. Indeed, the “what” question amounts to developing a scene model from a series of fixations. Each fixation is characterized by “location” and “content” (the image in a window around the fixation point). If X_i is the hyper vector denoting the fixation location and Y_i the hyper vector denoting the image around X_i , then by binding X_i to Y_i and summing up the hyper vectors for all fixations, we obtain $\sigma X_i \times Yi$, a scene hyper vector based on fixations. Similarly, the when question can be addressed with hyper vectors. If $V(t)$ and $M(t)$ are the visual and motor signals during an action over time t, then for any t, the binding of the hyper vector $V(t)$ with $M(t)$ gives a new hypervector which is an “instant”. Turning a sequence of such “instants” into a new hypervector leads to a history vector. The same can be said about the rest of the questions: for example, alignment problems amount to servoing of different kinds, where servoing is viewed as prediction of an action with a goal. All of the above constitute future research goals.

Here, we focus on a simple problem that will allow us to evaluate our approach in depth. Thus, we focus on the problem of 3D motion. First, we describe a framework for integrating multiple modalities such as perception and action together into a single space, to produce features for learning in a manner that’s desirable for active perception. Additionally, we show how the DVS can be used in such a framework. We begin with some background information about HBVs and what they are capable of. Next, we describe how recent work on HBVs can be used to effectively generate meaningful, semantically informed HBVs and integrate them together to form representations for sequences or sets of data. Following that, a method for encoding more complex signals such as images is discussed. We then describe recent work on DVS that is used to generate motion based perception, and how HBV representations of this can be made. Additionally, we describe a method for integrating action and perception into a single space and show how the resulting HBVs can be used as "memories" of prior actions and perceptions. We show results on the use of HBVs with DVS based information and how it compares with a more traditional vision approach to DVS signals to predict velocity and ego-motion. Finally, we discuss our results and how they can be used in future applications to further facilitate the use of HAP and form a more coherent path to better AI in robotics.

B.2 The Properties of Hyperdimensional Binary Vectors (HBVs)

Much of the inspiration behind the ideas discussed in this paper come from Kanerva's work on Hyperdimensional Computing [7]. The basic idea is to think about what sort of operations and information are computable and encodable with vectors of extreme dimensions. In the case of binary inputs, we find that there are several desirable properties inherent to the space of HBVs. We describe these in this section, in order to provide some necessary background.

Consider the case of a binary vector space of 10000 dimensions, containing a staggering number of unique vectors; exactly 2^{10000}. It's unlikely that any system will require so many unique vectors, as it would not even have enough time to enumerate all possibilities. Since the space forms a very high dimensional hypercube, with as many vertices as unique points, it is apparent that the distribution of points (vertices) from the perspective of any one point (vertex) is always the same. This is akin to how a cube looks the same from the perspective of any corner. Thus, binary vectors are a way to create "arbitrary" representational vectors for computers.

Consider the Hamming Distance (H) between two binary vectors: $|a \times b|$, where $\times$ is the bitwise XOR operation and magnitude is the number of 1's in the result. The number of components of disagreement between a and b is thus the distance between them. We can normalize this value to be agnostic to the dimensions of the vectors. If n is the dimension of the vectors, H can be normalized to the range of values between 0 and 1 by dividing the result by n, referred to as Normalized Hamming Distance (H_n). By assuming each binary vector is equally likely, it's clear that the average H_n between vectors is 0.5 ($5,000$ bits), with a standard deviation of 0.005 (50 bits), by a binomial distribution. The same is true of the distribution of distances from any one point to any other point. Yet, straying from the average distance of 0.5 is very difficult. As each increment 0.005 is a standard deviation, it is highly unusual for two random vectors to have a distance of 0.475 or 0.525, as this is a whole 5 standard deviations. Indeed, even a distance of 0.333 is far too unlikely to be random. As a result, HBVs are quite tolerant to noise in measurement. A distance deviating a few standard deviations from 0 may as well be the same vector in most applications. Likewise, randomly drawn vectors are nearly guaranteed to be close to a distance of 0.5. As a consequence, HBVs are very good candidates for encoding various forms of information into constant sized vectors.

Kanerva describes 3 particular operations that are well suited for encoding information:

1. The XOR operation: Since XOR is an involution when one operand is fixed, and associative and commutative, we have that $c \times a = (a \times b) \times a = b$. We can exactly recover a or b if we have one or the other, or approximately when noise is present.

2. The permutation Π: This permutes a vector x's components into a new order, by computing the product Π_x. If the permutation is randomly generated for a long binary vector, the new binary vector is very likely to have a $H_n \approx 0.5$. We can represent Π as a permutation of index locations 1 to n. The product simply swaps components of x to the order in Π.

3. The consensus sum, $c_+(A)$, over the set of vectors A, or simply $x + cy + cz$: This sum counts 1's and 0's component-wise across each element of A, and sets the component to the corresponding value with the bigger count. Ties, only possible in a sum of an even number of elements, can be broken by randomly choosing 0 or 1.

Other options exist for encoding, but these 3 are of particular concern to this chapter. Note that mapping by XOR or permuting preserves distances. For a given mapping a:

$$H(a \times x, a \times y) = |a \times x \times a \times y| = |a \times a \times x \times y| = |x \times y| \tag{B.1}$$

$$H(\Pi_x, \Pi_y) = |\Pi_x \times \Pi_y| = |\Pi(x \times y)| = |x \times y| \tag{B.2}$$

The permutation example, in particular, is due to permutations distributing across XOR and the fact that permutations due not change the number of 1's or 0's in the result of an XOR.

With the 3 operations above, we can now represent more complex data structures entirely with HBVs:

1. A set can be represented as follows: given $C_1, C_2, ..., C_n$ and a mapping of each element to HBVs $z_1, z_2, ..., z_n$, we encode the set as the XOR of each HBV, or

$$z = z_1 \times z_2 \times ... \times z_n \tag{B.3}$$

No matter the order of each C_i , the representation will always be the same, thus imitating a set. There are some limits to this representation. For example, while testing equivalency is simple (check if XOR is the 0 vector), testing membership and enumeration cannot be done without having another binary vector representation in place specifically for this. Furthermore, adding the same element has the effect of removing it, and care must be taken to prevent that from occurring.

2. Similarly, for an ordered pair (A, B), we can choose a random permutation Π and encode the pair as $c = \Pi(a \times b)$, where a and b are the HBVs associated with A and B. The permutation Π thus encodes the datatype of the ordered pair, as well. Given c, and knowing Π, and one of the pairs, the other can be found.

3. Sequences can be interpreted as a succession of ordered pairs. Thus, a sequence $C_1C_2...C_n$ is equivalent to

$$c = \Pi^{n-1}c_1 \times \Pi^{n-2}c_2 \times ... \times c_n \tag{B.4}$$

where c_i are the corresponding HBVs. The Π_i refers to a permutation that has applied itself to itself i times, before being applied to the HBV's components. Note that XORing two separate sequences will remove any patterns they both share. Furthermore, sequences can be shifted by permuting the vector again with Π to move further backwards (to add a new part of the sequence), or it can be shifted forward in time by applying Π^{-1}, or the de-permutation.

4. Data Records are referred to by Kanerva as a method for storing object-like properties. For example, a data record could consist of name, age, and sex. Each of these is given a random identifier r_i , and the data record format can be represented as a matrix $R = [r_1 r_2 ... r_n]$. To "bind" a value to each identifier, we can XOR the value with the corresponding identifier. Thus, given i values, v_i, a particular data record is represented as

$$R \times V = [r_1 r_2 ... r_n][v_1 v_2 ... v_n]^T = r_1 \times v_1 +_c r_2 \times v_2 +_c ... +_c r_n \times v_n \qquad \text{(B.5)}$$

   Where the bracket notation defines a "matrix" of vectors, with multiplication of matrices following the typical notation, but with XOR and consensus sum replacing multiplication and addition of elements. To isolate the value of a field, we simply compute $R \times r_i$, or $R \times v_i$ if we wish to find the data type a value is associated with. This is due to the fact that each term in the consensus sum will produce random noise (as distributing the XOR across the data record will yield randomness), but the term containing the value will be significant. You will get an approximately similar result; that is, the nearest neighbor will be, with high likelihood, the corresponding value or identifier.

With these simple structures, data records, sets and sequences can be combined to encode and represent all manner of data in a consistent way.

B.3 Numerical vs. Categorical Values

One of the simplest and most necessary forms of data are that of raw numerical values. How should the distance H between vectors representing numbers correlate to their objective value? One simple answer is to directly embed the linear relationship between numbers in a higher dimensional space, as a line or a curve in that space. This is clearly possible with binary vectors, though some care should be taken to ensure that these values make sense in their distances. For example, a given value X, the values $X-1$ and $X+1$ should be nearest neighbors to X, but on roughly opposite locations in space from X.

With categorical values, we are unsure of what the appropriate distance from them should be. For example, what is the distance between a and b? For this we must rely on distributional semantics, where the distributional properties of how these values occur and co-occur in data resolve their semantic meaning towards each other. Values that appear in similar distributions of data are taken to be closer in meaning. It is clear that both numerical and categorical data are important in AI. The real difficulty in representing them comes into play when we consider that they have to exist in the same space.

B.4 Representing Numerical and Categorical Data

In prior work [8], a method for encoding multiple modalities was developed to both create HBVs for observations of data, using distributional semantics, and give a framework for forming more complex sequences and sets of differing modalities, with potential for their own distributed semantics. Although this is not the particular subject of this paper, it is worth remarking on this process, as we use the numerical representations of it in our experiments.

Consider that you have multiple modalities, such as vision, audio, text, etc. Each of these has a finite number of unique values to represent an atomic piece of information. Consider the special case where the atomic piece of information does not actually have any distance associated with it. This is true for characters, as any meaning ascribed to them is purely distributional (how these values are distributed in relation to others). Pixel intensity and related measures all have inherent meaning in their values, directly tied to the distance between each value (e.g., red channel intensity of 55 vs. 123). Thus, we desire the ability to both encode distributional and non-distributional values in a framework for HBV representation of multiple modalities.

Take characters as an example for convenience, we can find distributionally meaningful HBVs for the space of characters by the method described in [8]. First, assign every character a randomly selected HBV. Then, obtain distributional counts of how often two characters may co-occur. The distributional counts can be naturally represented as a graph, where the vertices are characters, and the directed edges are co-occurrences. This is similar to systems such as word2vec [9] or GloVE [10]. Each character has a "mass", of how often it is seen, and an edge weight determined by how often that occurrence occurs. A good choice of HBVs for each character would make sure that characters that co-occur often are nearer than those that do not. This is referred to as a geometric interpretation of semantics. To find such vectors, we can simulate a model that characterizes energy in the current system of HBVs. Let $X^{(k)}$ be a matrix of current HBVs, with m rows/vertices and n columns/bit length. The problem is formulated as:

$$argmin(T(X^{(k)} + X)) \tag{B.6}$$

Here, T is a function that measures the total energy in the system described by a given matrix. Our goal is to find a perturbation of 1's and 0's referred to as X, that minimizes the energy measure by T. Then $X^{(k+1)} = X^{(x)} + X$ becomes our new set of vectors. We repeat this process until the energy is minimized. The energy is computed as:

$$T(A) = \Sigma\Sigma max(F_{conn}(A, i, j) + F_{prox}(A, i, j), 0) \tag{B.7}$$

This formula expresses the sum of all unresolved forces in the matrix A, given a graph of co-occurrences. The functions F_{conn} and F_{prox} represent the connective and proximal forces of the system. Together, they enumerate the force being applied on the bit at row i and column j to switch its value. The connective force for a vertex is the resultant force across all co-occurrence pairs of that vertex. An edge's weight (co-occurrence likelihood) dictates its force. The connective force is computed by the following, where M_i is the relative mass of vertex i and $W_{i,k}$ is the edge weight between vertex i and k (note that the weight is 0 if a connection does not exist):

$$F_{conn}(A, i, j) = \Sigma M_i W_{i,k} C_{conn}(A_{i,j}, A_{k,j}) \tag{B.8}$$

The proximal force is caused by the proximity of a vertex to a vertex that it shares a co- occurrence edge with. The force each vertex generates scales with its mass (occur-

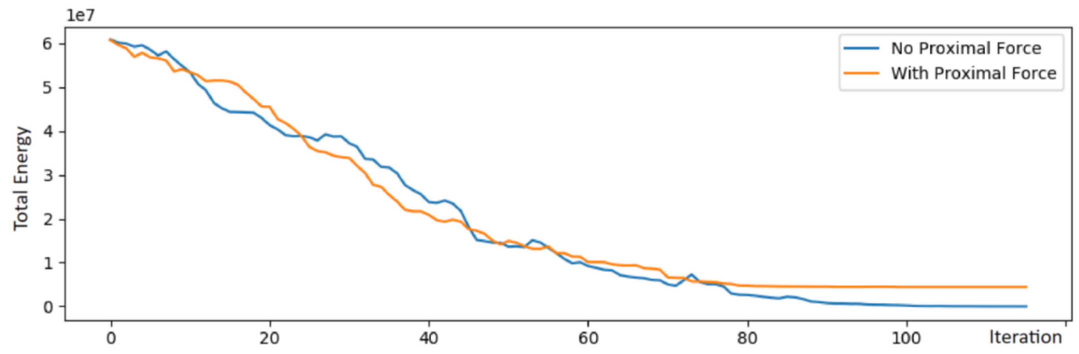


Figure B.1: *An example of how tension is minimized in a graph represented the co-occurrences of vertices. As the total energy in the system decreases with each iterations, the positions of the HBVs associated with each vertex are in geometrically "better" locations, relative to each other. When proximal force is turned off, the system is enabled to reach a singularity state, and thus, the tension will naturally reach 0.*

rence likelihood). The proximal force is computed by the following, where $H_n(a,b)$ is the normalized Hamming Distance between vertexes a and b:

$$F_{prox}(A,i,j) = \Sigma \frac{M_i M_k}{H_n(A_i, A_k)} C_{prox}(A_{i,j}, A_{k,j}) \tag{B.9}$$

The functions C_{conn} and C_{prox} are simply functions that determine whether the bit in question wants to change or not, given its current state and the state of its incident vertex

$$C_{conn}(a,b) = \begin{cases} -1 & \text{if } a = b \\ 1 & \text{if } a \neq b \end{cases} \quad C_{prox}(a,b) = \begin{cases} 1 & \text{if } a = b \\ -1 & \text{if } a \neq b \end{cases} \tag{B.10}$$

Techniques exist for minimization of T and are stochastic in nature, involving gradual flipping of bits to minimize energy, often using simulated annealing. As shown in B.1, the energy of the same system generated by random graphs of equal number of vertices with and without proximal force only reaches exact 0 when the proximal force is off.

B.5 Encoding images as HBVs

To be able to encode more complex structures as HBVs, such as images, care must be taken to preserve the meaningfulness of both the values of pixel intensity and location. How can this be done with the basic structures we can encode? First, we describe how pixel intensity is represented with HBVs. Consider the case of a single-channel, gray-scale image. We have 256 unique values across the space of intensities, all evenly spaced from one another. To reconstruct this with vectors, we first require 256 vectors and then require them to be spaced such that the nearest neighbors of a particular intensity vertex are the closest intensities in reality. Likewise, intensities that are further away in value should have a further H or Hn distance proportionally. A proportionally similar distance should be found for pixels in lesser or greater intensities. Essentially, the intensities, which are naturally visualized as a 1D line, are embedded into a higher-dimensional space as a curve or a line of vectors. We can visualize this by taking a particular vertex and connecting it to each other vertex, requiring that the edge weights decrease proportionally to the difference in intensity between the vertices, evenly spaced out. Applied to every vertex, we arrive at the realization that a fully connected graph (apart from vertices connected to themselves) is

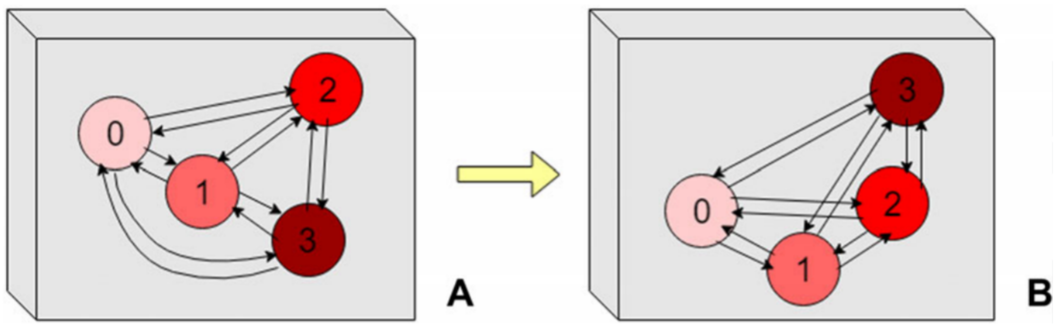


Figure B.2: *A visualization of intensity minimization. We start with random vectors for four intensities ranging from 0 to 3. A complete graph is formed, connecting each vertex to each other, with decreasing weight in proportion to the intensity difference. When minimization is performed, the result forms a line or semicircle of appropriate distances.*

required. Thus, every vertex should have a vector that satisfies the properties of the intensity scale. Fig. B.2 shows how that this is visualized on a small scale. When this graph is minimized with the technique described in the previous section, the result forms a distance matrix much like the one shown in Fig. B.3. As we can see, the distances increase away from the diagonal entry, much like intensities do in a single channel, from the perspective of a particular intensity value.

With representations for the intensities, we can now focus on representing location. Permutations hold the positional semantics for sequences of HBVs; a particular permutation is attached to a sequence of a single data type. This is movement through sequences in a one directional sense. For 2D data such as images, we require two permutation operations, one for the row and one for the column location. For each location, we permute the intensity representations appropriately. Consider a single such pixel; we can place it

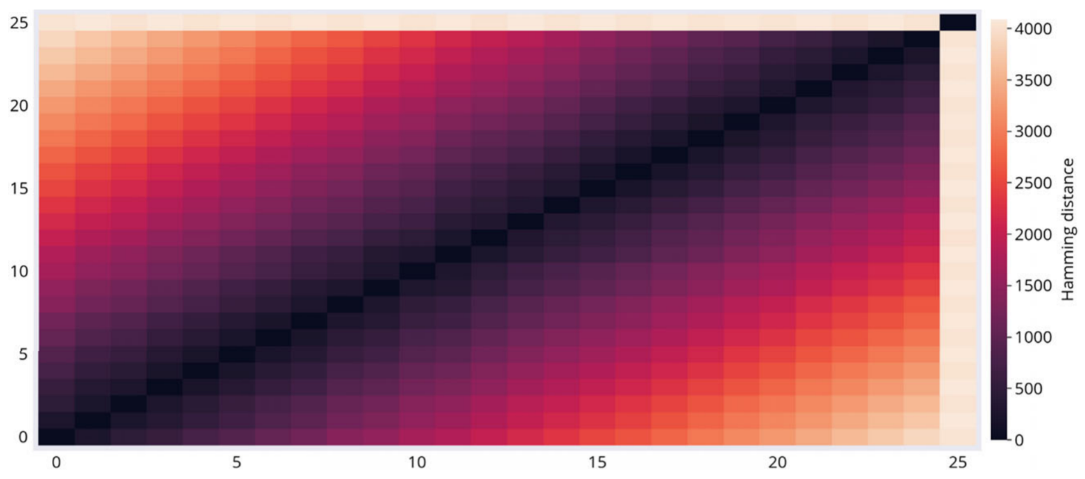


Figure B.3: *Hamming distance visualization of intensities. Visualization of the Hamming distance between intensity values in the range of 0 to 25. Note the increasing distances away from the diagonal, as one would expect. For DVS or images, a full 256-intensity range can be easily developed with the same properties.*

in the proper location in an image by permutations before XORing with other pixels—the concept of which is shown in Fig. B.4. As a small example, consider a 3 by 3 image I. Let $_{Iij}$ be the pixel intensity representation at row i and column j. Then, the HBV representation of the image is constructed as follows

$$(X00 * CX_{01} * C^2X_{02}) * R(X_{10} * CX_{11} * C^2X12) * R^2(X_{20} * CX21 * C^2X22) \quad \text{(B.11)}$$

This simple but powerful formulation allows us to encode an image of arbitrary dimensions in meaningful ways. Much like with sequence, permutations of the resulting HBV will shift the image in space accordingly, and entire sections of the image can be removed or concatenated with others by first XORing with the section to remove in the full image and XORing with a section to add, as shown in Fig. B.5.

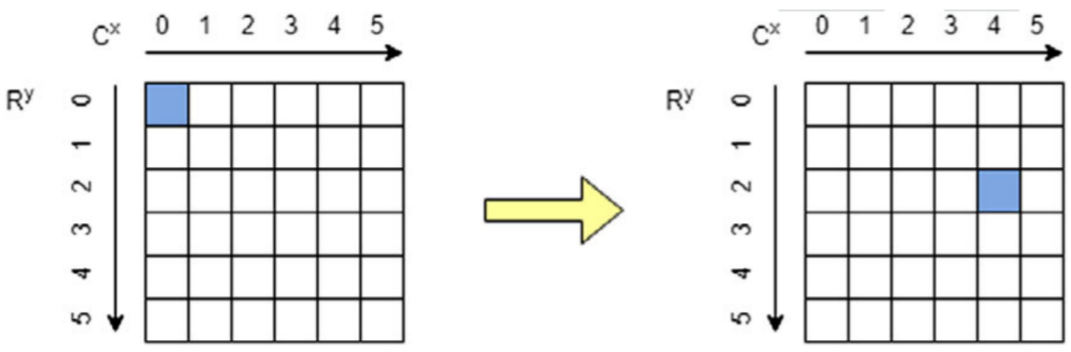


Figure B.4: *Moving pixels spatially. A pixel p is relocated from the origin position to row 2 and column 4 by performing a permutation* R^2C^4p

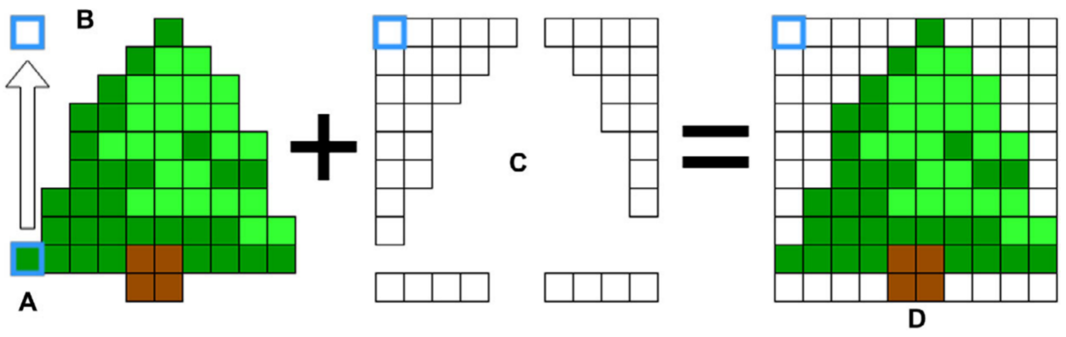


Figure B.5: *Composing image encodings. A simple example of how an image encoding can be manipulated. The reference pixel can be arbitrarily moved through permutations as shown in (A) by permuting the current encoding appropriately; in the case of (B), this is a permutation of* R^{-8}*. A new image (C), aligned on a similar reference pixel (the origin here); the two images can be composed to construct an arbitrary image (D).*

B.6 Creating memories with HBVs

With the ability of HBVs to combine with other HBVs to encode more and more information but in the same vector length, we have a natural method to create semantically significant and informed memories. As an example, a series of images can be "remembered" by forming a data record with HBVs of each image, where a particular vector is taken to signify the location in time of the image. When presented with a particular image, its existence inside of a memory can be checked by simply XORing the memory with the image. The nearest neighbor of the result will be the location in time of the image. Likewise, when interested in the image at a particular location, the same process will reveal what the nearest matching image will be.

B.7 Experiments

B.7.1 Using neuromorphic visual information

We extracted the motion information from the event timestamps by building a "time image": In a given time slice of the (x, y, t) space (that is, for a small interval of time), the events are projected on the image plane and the timestamps of the events that fall on the same pixel are averaged out, so the pixel intensity is a function of time t. This representation has several benefits: It is easy to compute, is robust toward noise, allows for asynchronous data analysis (because the time image can be computed at any moment in time), and preserves the motion information stored within the event stream. An example of the time image, together with the corresponding "classical" frame, can be seen in Fig. B.6.

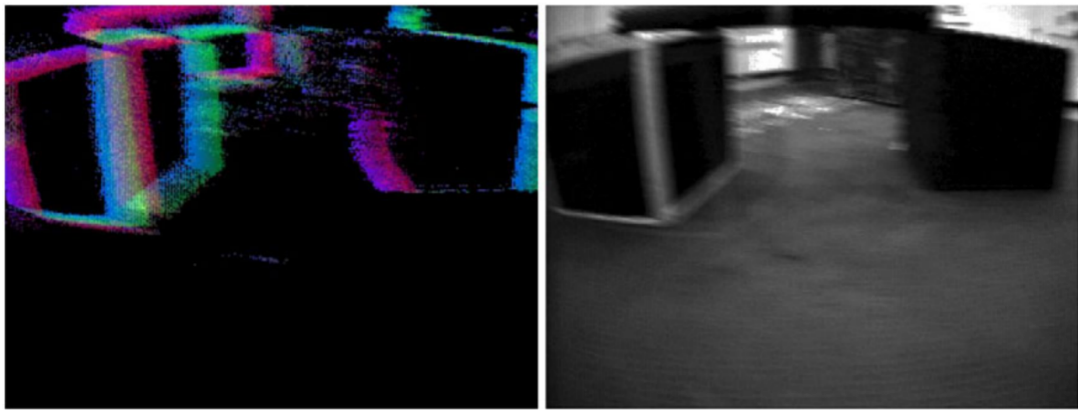


Figure B.6: *An example of DVS data visualization. On the left, a time image is shown, in which green is the average timestamp, and the positive/negative per pixel event counts are shown in red/blue. On the right, the corresponding grayscale image of the scene is shown. Note the motion blur on the classical frame.*

B.7.2 Learning with HBVs

To test how well visual data can be encoded by HBVs when compared to classical techniques, we set up an experiment where a classical convolutional neural network (CNN) architecture, inspired by NVIDIA's PilotNet, was trained on the DVS event stream to predict velocities. The input layer of the CNN accepted time images as described in the previous section, essentially interpreted as a single-channel image. Both the event stream and velocities were collected via an autonomous robotic platform that was tracked by the VICON Motion Capture System. The magnitudes of the velocities were in the order of 1 m/s, and the DVS camera experienced 6D motion. With this setup, the CNN was able to achieve an accuracy of up to 7 cm/s, amounting to 7% of the velocity magnitude, despite the sparseness of the event data.

Likewise, the time image data were also converted into HBV encodings in a fashion similar to how image codes are constructed. This was achieved by first constructing an intensity space consisting of 255 HBVs of length 8000, each corresponding to one of the 255 intensity values of the time image and minimizing it to generate appropriate HBV representations. When visualized, this space is similar to what is shown in Fig. B.3 with 25 intensity values. We only used 255 HBVs because the intensity value of 0 indicates no event and thus does not need to be encoded into the HBV. These representations are then used to generate sequences encoding the pixel intensities and their locations in each time image into a single HBV. Because of the fact that bits in HBVs do not necessarily correlate to neighboring bits, it does not make sense to use a similar, CNN-based network for HBVs as we did in our first experiment. As a result, a fully connected, six layered neural network was used with a similar number of parameters. When trained, HBV-based learning achieved an error of about 9 cm/s. Although slightly worse than the CNN, this result is impressive because the size of the input was naturally compressed as HBVs, from the DVS's resolution of $346 * 260$ (about 90,000 pixels) to 8000 bits. Inspired by these results, we then investigated the HBV's ability to encode sequences of frames as data records and binding the velocity vectors to the frame vectors.

B.7.3 Perception to action binding with HBVs

The HBV vectors allow any amount of data of any type to be represented in a fixed length of bits, up to the informational capacity provided by the binary space. We would expect HBVs to be able to create data records from different modalities to bind them together in a single representational space—ideally, binding perception to action as a single data record. We constructed an experiment where, given the visual input, the system was able to remember the action it needed to take in the form of a velocity vector.

First, we describe how to represent a 3D velocity vector as an HBV. We construct HBV representations of each component individually, that is, we run vector space minimization three times with different starting seeds for random HBVs. We then create a data record from the three spaces for a single velocity. To do this, we must select three random binary vectors as identifiers for the X, Y, and Z components of velocity to construct the data record with. We will refer to these as X, Y, and Z. Because these are random, they are nearly unrelated, with distances close to 4000 bits from each other (to be consistent, we use 8000-bit vectors in all our experiments). When component values are bound to the identifiers, they each create three separate uncorrelated HBVs. This velocity data record is extended with an entry for the associated time image HBV encoding. Likewise, this encoding is bound to a randomly selected identifier for time images, referred to as T. The resultant data record thus exists in an action perception space and can be considered a memory.

To create many such memories, we take a DVS recording with corresponding motion ground truth and create a memory of each time image and velocity. Our experiment: Given a time image, find the correctly associated velocity using only memories. To isolate the time image in the data record of the memory, we XOR T with the memory m and find the nearest neighbor to the input time image encoding. Afterward, we XOR the candidate memory with X, Y, and Z to retrieve individual velocity components. In all our experiments, we achieved 100% accuracy. This means that the information capacity of the data record with 8000 bits far exceeds the requirements to store the time image and velocities encoded within.

B.7.4 Information capacity of HBVs

A natural next experiment would be to see how much information can be fit into 8000-bit vectors. Because each time image contains quite a bit of information, we chose to encode entire sequences of time images in a single data record, with randomly generated HBV identifiers T_i tied to each frame position in the sequence. We then tested the ability of data records to localize the given image in the memories containing only time image sequences.

Fig. B.7 shows our findings. We conducted the experiments for sequences of up to 700 frames in length and measured the Hamming distance of the input frame HBV to the closest match in memory of sequences. Naturally, the longer the sequence, the more information is contained within every data record. Thus, we expect that the Hamming distances will grow closer and closer to 4000, which corresponds to completely uncorrelated vectors. Our results confirm this trend and show that we can safely code up to around 200 frames without worry of false positives. However, the likelihood of errors is small even after this point. In conclusion, a single 8000-bit HBV can provide for huge information density, without needing any learning.

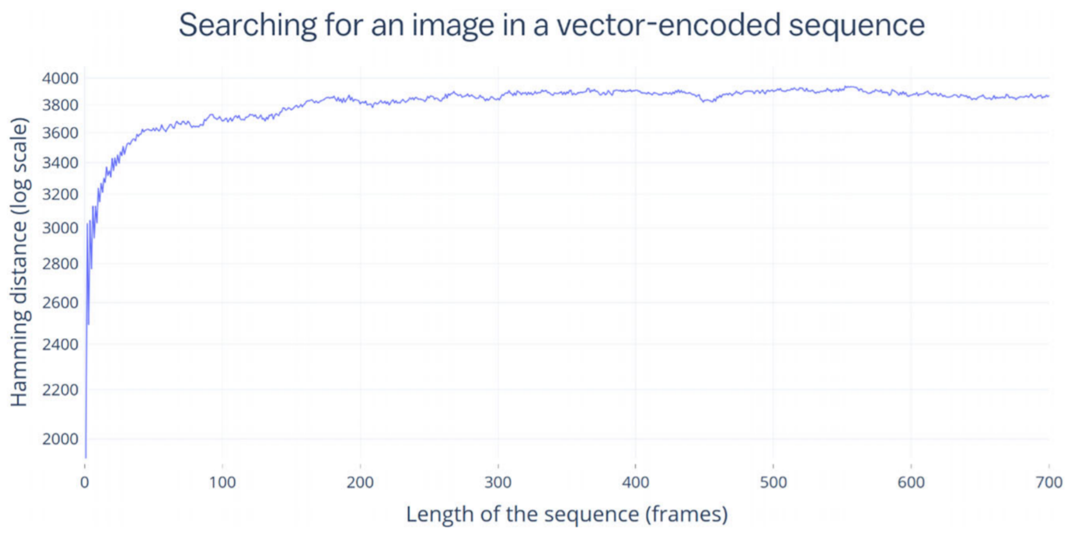


Figure B.7: *Hamming distance decay in data records. The decay in nearest neighbor matches of Hamming distance as the number of time images in a memory increases. As more frames of motion are put into the encoding, more noise is introduced, making the deviation caused by a match less and less statistically significant. At 700 frames, we are still about 3 to 4 SDs of likelihood away from random vectors.*

B.7.4.1 Theoretical limits on capacity of HBVs

Although we have empirical results on how well information can be encoded into HBVs, we can arrive at more practical bounds for this in relation to the accuracy of recall. Regardless of the length of the HBVs, the odds of a particular bit being 0 or 1 is 0.5, assuming that the vectors are randomly generated. Suppose we have performed a consensus sum on a number of data points using random vectors as a basis for identifying each data point (i.e., a data record). The result can be thought of as a memory consisting of these data points. Let m be the memory, then (with the right-most side as shorthand notation)

$$m = x_1 * a +_c x_2 * b +_c x_3 * c ... = x_1 a + x_2 b + x_3 c + ... \tag{B.12}$$

When we test for the presence of a particular value d, we first compute $m * d$. The result should be close to a particular x_i, if d is present. Naturally, to determine this, we compute the distance H. This involves another XOR with each x_i individually. Consider the following example:

$$|m * d * x_1| = |x_1 a d x_1 + x_2 b d x_1 + x_3 c d x_1 + ...| = |ad + x_1 x_2 b d + x_1 x_3 c d + ...| \tag{B.13}$$

When computing the H_n distance, the above becomes the probability of each bit being 1 in this vector. Because we are computing a consensus sum, the above is equivalent to tossing a coin the same number of times as you have terms in the sum and setting the consensus to whichever side landed more often. This is true because we know that, if the vectors are random, then the XOR of two random vectors yields a random vector. However, if $a = d$, then the term $ad = 0$. This would create a bias; one of our coin tosses will now always be the same. More generally, let the probability of 1 for the term ad be p. This bias is what causes our consensus sum to give a H_n that is different in a statistically significant way than the expected value of 0.5. Given n vectors in a consensus sum and p, the probability for a bit in $|m * d * x_1|$ being 1 is

$$(1-p) \sum_{k=n/2}^{n-1} \frac{(n-1)!}{2^{n-1} k! \, (n-k-1)!} + p \sum_{k=n/2-1}^{n-1} \frac{(n-1)!}{2^{n-1} k! \, (n-k-1)!} \tag{B.14}$$

B.7.5 Sensorimotor representation through binding visual and motor information

To demonstrate an example application of our framework, we applied it to the task of ego-motion estimation in the autonomous vehicle setting. We used the multivehicle stereo event camera (MVSEC) dataset for our experiments. The MVSEC dataset features DAVIS240 event recordings together with ground truth velocity, taken from a car driving during the day and night. We formed a separate memory for each degree of freedom using the first 15 s of the "outdoor day 1" sequence and then demonstrated the performance on the rest of the sequences (two during the day and three during the night). We compute the memory as follows, for each degree of freedom

$$m = v_1(a_1 + a_2 + ...) + v_2(b_1 + b_2 + ...) + ... + v_n(z_1 + z_2 + ...) \tag{B.15}$$

Here, the vectors v_i are basis vectors for our velocity space, that is, every vector represents a particular range of velocities with a certain step. In our experiments, 0.001 was used as the step. The distinct letters in $a_i, ..., zi$ represent groups of images by their ground truth velocity class. Conceptually, the memory m is just a consensus sum of every image vector bound to every corresponding 1D velocity represented as an HBV. We then compute the probability of the vector d representing an incoming image being associated with each of the basis velocity vectors v_i and choose the v_i yielding the highest probability, equivalent to the smallest H between mv_i and d.

Bibliography

[1] Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. HATS: histograms of averaged time surfaces for robust event-based object classification. *CoRR*, abs/1803.07913, 2018.

[2] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128 x 128 at 120db 15 micros latency asynchronous temporal contrast vision sensor. *Solid-State Circuits, IEEE Journal of*, 43(2):566–576, 2008.

[3] C. Posch, D. Matolin, and R. Wohlgenannt. A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds. *Solid-State Circuits, IEEE Journal of*, 46(1):259–275, 2011.

[4] C. Brandli, R. Berner, M. Yang, S. Liu, and T. Delbruck. A 240 × 180 130 db 3 µs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, Oct 2014.

[5] B. Son, Y. Suh, S. Kim, H. Jung, J. Kim, C. Shin, K. Park, K. Lee, J. Park, J. Woo, Y. Roh, H. Lee, Y. Wang, I. Ovsiannikov, and H. Ryu. 4.1 a 640×480 dynamic vision sensor with a 9µm pixel and 300meps address-event representation. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 66–67, Feb 2017.

[6] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *ArXiv*, abs/1904.08405, 2019.

[7] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[8] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1052–1067, June 2007.

[9] Jan J Koenderink. Optic flow. *Vision research*, 26(1):161–179, 1986.

[10] Hugh Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[11] Edward H Adelson and James R Bergen. Spatiotemporal energy models for the perception of motion. *J. Opt. Soc. Am. A*, 2(2), 1985.

[12] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, June 2012.

[13] R. Benosman, C. Clercq, X. Lagorce, Sio-Hoi Ieng, and C. Bartolozzi. Event-based visual flow. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(2):407–417, 2014.

[14] Xavier Clady, Charles Clercq, Sio-Hoi Ieng, Fouzhan Houseini, Marco Randazzo, Lorenzo Natale, Chiara Bartolozzi, and Ryad B. Benosman. Asynchronous visual event-based time-to-contact. In *Front. Neurosci.*, 2014.

[15] Elias Mueggler, Christian Forster, Nathan Baumli, Guillermo Gallego, and Davide Scaramuzza. Lifetime estimation of events from dynamic vision sensors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4874–4881. IEEE, 2015.

[16] Garrick Orchard, Ryad Benosman, Ralph Etienne-Cummings, and Nitish V Thakor. A spiking neural network architecture for visual motion estimation. In *Biomedical Circuits and Systems Conference (BioCAS), 2013 IEEE*, pages 298–301. IEEE, 2013.

[17] Min Liu and Tobi Delbruck. Block-matching optical flow for dynamic vision sensors: Algorithm and fpga implementation. In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pages 1–4. IEEE, 2017.

[18] Francisco Barranco, Cornelia Fermüller, and Yiannis Aloimonos. Contour motion estimation for asynchronous event-driven cameras. *Proceedings of the IEEE*, 102(10):1537–1556, 2014.

[19] Anton Mitrokhin, Cornelia Fermuller, Chethan Parameshwara, and Yiannis Aloimonos. Event-based moving object detection and tracking. *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2018.

[20] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. In *CVPR*, 2018.

[21] D. Tedaldi, G. Gallego, E. Mueggler, and D. Scaramuzza. Feature detection and tracking with the dynamic and active-pixel vision sensor (davis). In *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pages 1–7, June 2016.

[22] A. Z. Zhu, N. Atanasov, and K. Daniilidis. Event-based feature tracking with probabilistic data association. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4465–4470, May 2017.

[23] Jacques Manderscheid, Amos Sironi, Nicolas Bourdis, Davide Migliore, and Vincent Lepetit. Speed invariant time surface for learning to detect corner points with event-based cameras. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[24] Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E Shi, and Ryad B Benosman. Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1346–1359, 2016.

[25] Thusitha N Chandrapala and Bertram E Shi. Invariant feature extraction from event based stimuli. In *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 1–6. IEEE, 2016.

[26] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Ev-flownet: Self-supervised optical flow estimation for event-based cameras, 2018.

[27] Chengxi Ye, Anton Mitrokhin, Cornelia Fermüller, James A. Yorke, and Yiannis Aloimonos. Unsupervised learning of dense optical flow, depth and egomotion from sparse event data. *CoRR*, abs/1809.08625v2, 2018.

[28] A. Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, and K. Daniilidis. The multivehicle stereo event camera dataset: An event camera dataset for 3d perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, July 2018.

[29] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. *CoRR*, abs/1812.08156, 2018.

[30] Francisco Barranco, Ching L Teo, Cornelia Fermüller, and Yiannis Aloimonos. Contour detection and characterization for asynchronous event sensors. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 486–494, 2015.

[31] Anton Mitrokhin, Chengxi Ye, Cornelia Fermüller, Yiannis Aloimonos, and Tobi Delbrück. EV-IMO: motion segmentation dataset and learning pipeline for event cameras. *CoRR*, abs/1903.07520, 2019.

[32] Ashutosh Saxena, Sung H Chung, and Andrew Y Ng. Learning depth from single monocular images. In *Advances in neural information processing systems*, pages 1161–1168, 2006.

[33] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[34] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *European Conference on Computer Vision*, pages 740–756. Springer, 2016.

[35] Junyuan Xie, Ross Girshick, and Ali Farhadi. Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In *European Conference on Computer Vision*, pages 842–857. Springer, 2016.

[36] Yi Zhou, Guillermo Gallego, Henri Rebecq, Laurent Kneip, Hongdong Li, and Davide Scaramuzza. Semi-dense 3d reconstruction with a stereo event camera. *European Conference on Computer Vision(ECCV)*, 2018.

[37] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised Event-based Learning of Optical Flow, Depth, and Egomotion. *arXiv e-prints*, page arXiv:1812.08156, Dec 2018.

[38] Guillermo Gallego Henri Rebecq and Davide Scaramuzza. Emvs: Event-based multi-view stereo. In E. R. Hancock R. C. Wilson and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 63.1–63.11, September 2016.

[39] Hanme Kim, Stefan Leutenegger, and Andrew J. Davison. Real-time 3d reconstruction and 6-dof tracking with an event camera. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 349–364, Cham, 2016. Springer International Publishing.

[40] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018.

[41] Deqing Sun, Erik B Sudderth, and Michael J Black. Layered segmentation and optical flow estimation over time. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1768–1775. IEEE, 2012.

[42] J-M Odobez and Patrick Bouthemy. Mrf-based motion segmentation exploiting a 2d motion model robust estimation. In *Proceedings., International Conference on Image Processing*, volume 3, pages 628–631. IEEE, 1995.

[43] William B. Thompson and Ting-Chuen Pong. Detecting moving objects. *International Journal of Computer Vision*, 4(1):39–57, Jan 1990.

[44] René Vidal, Yi Ma, and Shankar Sastry. Generalized principal component analysis (gpca). In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–I. IEEE, 2003.

[45] Michal Irani and P Anandan. A unified approach to moving object detection in 2d and 3d scenes. *IEEE transactions on pattern analysis and machine intelligence*, 20(6):577–589, 1998.

[46] Philip HS Torr. Geometric motion segmentation and model selection. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 356(1740):1321–1340, 1998.

[47] Abhijit S Ogale, Cornelia Fermuller, and Yiannis Aloimonos. Motion segmentation using occlusions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):988–992, 2005.

[48] Katerina Fragkiadaki, Geng Zhang, and Jianbo Shi. Video segmentation by tracing discontinuities in a trajectory embedding. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1846–1853. IEEE, 2012.

[49] Katerina Fragkiadaki, Pablo Arbelaez, Panna Felsen, and Jitendra Malik. Learning to segment moving objects in videos. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[50] Pia Bideau, Aruni RoyChowdhury, Rakesh R Menon, and Erik Learned-Miller. The best of both worlds: combining cnns and geometric constraints for hierarchical motion segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 508–517, 2018.

[51] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User Interface Software and Technology*, pages 559–568, 2011.

[52] Guillermo Gallego and Davide Scaramuzza. Accurate angular velocity estimation with an event camera. *IEEE Robotics and Automation Letters*, 2(2):632–639, 2017.

[53] Qualcomm Flight$^{\text{TM}}$. `https://developer.qualcomm.com/hardware/qualcomm-flight`, 2018.

[54] Kanerva and Pentti. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1, 06 2009.

[55] Robert C. Bolles, H. Harlyn Baker, and David H. Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision*, 1(1):7–55, Mar 1987.

[56] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 5105–5114, USA, 2017. Curran Associates Inc.

[57] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ArXiv*, abs/1801.07829, 2018.

[58] A. Szlam J. Bruna, W. Zaremba and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv:1312.6203v3*, 2017.

[59] Pierre Vandergheynst Michaël Defferrard, Xavier Bresson. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems 29 (2016)*, 2017.

[60] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[61] Yusuke Sekikawa, Kosuke Hara, and Hideo Saito. Eventnet: Asynchronous recursive event processing. *CoRR*, abs/1812.07045, 2018.

[62] Hui Ji and C. Fermuller. A 3d shape constraint on video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(6):1018–1023, 2006.

[63] Kalle Åström, Roberto Cipolla, and Peter Giblin. Generalised epipolar constraints. *International Journal of Computer Vision*, 33:51–72, 09 1999.

[64] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos. Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception. *Science Robotics*, 4(30), 2019.