Tampere University

Shashank Sahay

LOSSLESS COMPRESSION OF EVENT DATA AND OPTICAL FLOW IMAGES FROM EVENT CAMERAS

M. Sc. Thesis
Faculty of Information Technology and Communication Sciences
Dr. Ioan Tabus
Dr. Tapio Eloma
May 2023

ABSTRACT

Shashank Sahay: Lossless compression of event data and optical flow images from event cameras
M.Sc. Thesis
Tampere University
Master's Degree Programme in Computing Sciences
May 2023

This master's thesis investigates the problem of lossless compression of event data and optical flow images from event cameras. Event cameras are novel sensors capable of capturing high-speed motion with low latency and power consumptions. However, their high data rate and continuous data flow pose challenges in terms of storage and transmission. The proposed compression methods leverage the sparsity and temporal correlation present in event data and optical flow images to achieve high compression ratios while preserving the information content. The effectiveness of the proposed methods is evaluated using real-world datasets and compared with existing compression techniques. Optical flow images are generated using a pre-trained neural network and compressed thereafter using image codecs. Methods like scaling and quantization prove vital in doing the same. The results demonstrate the best methods to compress and transmit flow images and event data and have potential applications in fields such as autonomous driving, robotic machinery, augmented reality and motion compensation.

PREFACE

I would like to express my deepest gratitude to my thesis supervisor, Dr. Ioan Tabus, for his unwavering support, invaluable guidance, and constructive feedback throughout the entire process of writing this thesis. Without his help and encouragement, this thesis would not have been possible.

I would also like to thank my family for their unwavering support and encouragement during my graduate studies. Their love and patience have been a constant source of motivation for me.

My sincere thanks also go to my friends and colleagues for their helpful insights and support, which have been essential to my research. I am especially grateful to Rashi Verma for her dedication and constant encouragement during difficult times.

Thank you all for your support and encouragement, and for helping me achieve this significant milestone in my academic journey.

Tampere, 25 April 2023

Shashank Sahay

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

3D	Three-Dimensional
AEE	Average Endpoint Error
AEDAT	Address Event Data
AER	Address Event Representation
AP	Address Prior
APS	Active Pixel Sensor
AR	Augmented Reality
BWT	Burrows-Wheeler Transform
DAVIS	Dynamic and Active-pixel Vision Sensor
DB	Decibel
DVS	Dynamic Vision Sensor
FIRE	Fast Integer Regression
HDR	High Dynamic Range
JPEG	Joint Photographic Experts Group
LOCO	Low Complexity Lossless Compression for Images
LZ77	Lempel-Ziv-1977
LZMA	Lempel-Ziv-Markov-Chain Algorithm
MCP	Motion Compensated Prediction
MTF	Move-To-Front Transform
MVSEC	Multi Vehicle Stereo Event Camera
NAL	Network Abstraction Layer
PNG	Portable Network Graphics
PPMD	Prediction by Partial Matching
RELU	Rectified Linear Activation Unit

RGB	Red, Green, Blue
RLE	Run-Length Encoding
RTP/IP	Real Time Transport Protocol
TALVEN	Time Aggregation based Lossless Video Encoding for Neuromorphic vision Sensor Data
TANH	Hyperbolic Tangent Activation Function
TCP/IP	Transmission Control Protocol/Internet Protocol
TP	Time Prior
VR	Virtual Reality
ZLIB	Zeta Library

1. INTRODUCTION

Event cameras, also known as Dynamic Vision Sensors (DVS), have gained significant attention in recent years owing to their high temporal resolution, low power consumption, and robustness to motion blur [1]. These properties make them well-suited for a lot of applications, such as robotics, surveillance, and self-driving vehicles. However, the high data rate generated by event cameras poses a significant challenge for storage, transmission, and processing.

This thesis aims to address the challenge of event camera data compression by exploring various compression techniques and their effectiveness on event camera data. The goal of this research is to research a compression method that can significantly reduce the data rate of event cameras while preserving the important visual information. Event camera data is foundationally different from traditional frame-based image data, since it is a stream of asynchronous events, each representing a change in the brightness of a pixel in the image [2]. This difference in data representation poses a significant challenge for compression, as traditional compression methods are not well-suited for event data.

In this thesis, we will investigate different lossless compression techniques, to compress event camera data. We will also explore optical flow, it's applications and the use of machine learning to generate optical flows for the event camera data.

The effectiveness of the proposed compression methods will be evaluated using compression ratio and comparison against similar experiments by others [1]. The aim is to attain a high level of compression ratio while maintaining a low reconstruction error and high visual quality. The results of this research will have significant implications for event camera applications, as it will enable the efficient storage, transmission, and processing of event camera data.

In summary, this thesis aims to investigate various compression techniques for event camera data and develop a method that can efficiently reduce the data rate while preserving the important visual information. Furthermore, it will dis-

cuss optical flow, it’s applications and finally provide compression methods for the optical flows as well.

2. BACKGROUND

2.1 Event Cameras

An event camera, also called a neuromorphic camera or dynamic vision sensor, is of much interest to the scientific community. It differs from a traditional camera in the sense that it does not capture two dimensional images. Instead, it captures three-dimensional events. This is done by using a sensor that can detect variations in the intensity of the received light at each pixel location. On the occurrence of an event, the sensor records pixel location and the polarity of the intensity change. Using this property makes event cameras a great tool to use in applications involving moving objects. According to Khan et. Al,[2] event cameras mirror the visual processing properties of alive organisms. This means that they capture only scene reflectance changes. As mentioned by Guillermo et. Al, [1] event cameras are asynchronous sensors that will change the way we think of acquiring visual information. Because they gather light data in accordance with the movements of the scene being observed, compared to conventional cameras, event cameras possess some advantageous characteristics such as superior temporal resolution (measured in microseconds), a wide dynamic range (about 80 dB higher compared to traditional cameras), low power usage, and high pixel bandwidth. As a result, they can be used to minimize motion blur.

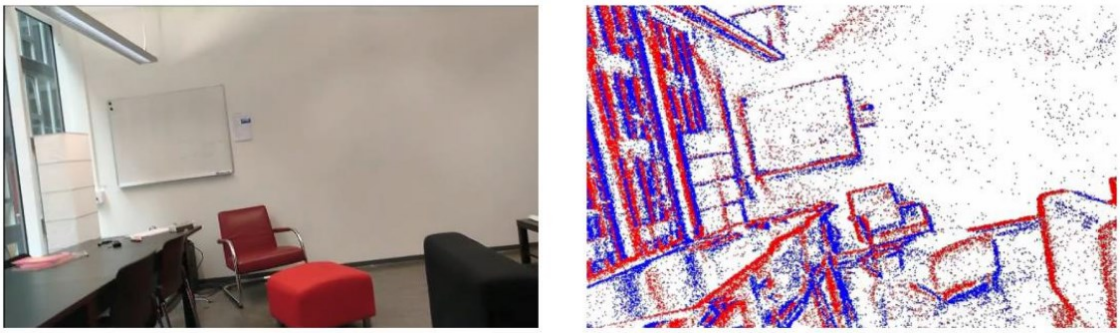


Figure 1. *Comparison of frames obtained from an event camera (right) to a conventional camera (left) [1]*

Based on Zhu et al., [3] whenever there is a modification in a pixel, the dynamic vision sensor produces an event denoted as e = (x, y, t, p). This event includes

the location coordinates of the pixel where the intensity modification occurred (x and y), the exact time when the change in brightness occurred (measured in microseconds and referred to as t), and the polarity of the change (referred to as p) which indicates the pixel becoming darker (0) or brighter (1). Numerous developments have been made in the realm of deep-learning methodologies and applications for frame-based jobs, which have been investigated in the context of an event-based imaging system as well. Event cameras have also been employed for tasks such as detecting objects, estimating visual odometry, predicting optical flow and depth. Even though event-based cameras demonstrate their superiority in challenging conditions, under normal circumstances, RGB cameras are still superior in terms of performance. [4].

DVS have many extremely useful applications and can change how we approach problems in the future. This following section presents some of these examples.

2.2 Application of DVS

We survey below a range of applications of the event cameras, that benefited most of the superior time resolution.

1. Robotics – Dynamic vision sensors can provide real-time visual data for robotic applications such as navigation, object detection and object tracking. Utilizing the extremely high temporal resolution of DVS, robots can have much better response times to stimuli.

2. Surveillance – DVS can be used for monitoring and security purposes in various settings such as airports, public places including malls, parks, etc and even private properties.

3. Automotive – By far the most promising application for DVS in the industry currently is with self-driving automotives. DVS can be of great use in driver assistance systems and autonomous vehicles to provide real-time visual data for obstacle/pedestrian detection, navigation and also lane keeping.

4. Augmented and Virtual reality – DVS when attached to a user’s head can help track and respond to the user’s movement, allowing for a more immersive experience in AR and VR applications.

5. Industrial Automation – Dynamic Vision sensors can also be used in industrial settings to track and inspect products on an assembly line, monitor equipment for maintenance and safety, and assist in automating manual tasks such as welding and painting.

2.3 Advantages of Event Cameras

Standard cameras can be surpassed by event cameras in terms of various potential benefits.:

High Temporal Resolution: Dynamic Vision Sensors (DVS) have high temporal resolution because they capture visual data as a stream of events rather than traditional frame-based images. Each event represents an alteration in the intensity of a pixel in the image, rather than a snapshot of the entire image. The high temporal resolution of DVS is achieved by utilizing silicon photodiodes which are sensitive to variations in light intensity instead of absolute light intensity. These photodiodes can detect changes in light intensity with extremely high temporal resolution, on the “order of microseconds” [37]. The high temporal resolution of DVS facilitates fast and accurate visual data acquisition in dynamic environments, making them fitting for real-world applications including robotics, surveillance, and autonomous vehicles [5].

Low Latency: In an event camera, every pixel operates autonomously, and there is no requirement to pause for a “global frame exposure time” [6]. The moment a change is identified, it is transferred immediately. Consequently, event cameras have very low latency, approximately “10 μs” in a laboratory setting, and less than a “millisecond” in practical scenarios [1].

Low Power: Since event cameras only communicate brightness change and remove surplus data, power is used only to handle the changing pixels. Typically, cameras consume roughly 10 mW of power, although there are some prototypes that require less than 10 μW. When an event camera is embedded in a system and directly connected to a processor, the overall power consumption (including sensing and processing) has been observed to be 100 mW or lower [7].

High Dynamic Range (HDR): Event cameras possess an extremely “high dynamic range of over 120 dB, surpassing the 60 dB of high-quality frame-based cameras” [1]. As a result, they can capture visual information across a vast range of illumination levels, from moonlight to daylight [2]. The logarithmic scale operation of photoreceptors in the pixels and the independent operation of each pixel contribute to this capability, allowing event cameras to adapt to stimuli that are very dark or very bright, akin to biological retinas.

Robustness to motion blur. DVS are not affected by motion blur, which can be a problem in traditional cameras while capturing fast moving objects. Since DVS cameras solely record an event in the event of a modification in the brightness of the scene, fast-moving objects will generate a large number of events, allowing the sensor to accurately track the object's movement. In traditional frame-based cameras, fast-moving objects will result in motion blur, as the camera takes a single snapshot of the scene, and the object will have moved during the exposure time.

2.4 Datasets and data format

For conducting experiments to compress event camera data, many different datasets are available. For this thesis, the PKU-DVS dataset and the DAVIS dataset were chosen. The former has been developed by National Engineering Laboratory for Video Technology (NELVT), Peking University [20]. This dataset consists of 2 different classes of data: indoor and outdoor. They have 7 and 6 datasets, respectively which capture scenes with moving objects. Some examples of this are pedestrians, a moving football, traffic etc. The latter is a public dataset which provides a set of high-speed video sequences with pixel-level ground truth annotations, captured under a variety of lighting conditions and scene dynamics. The dataset includes different types of events, such as object motion, brightness changes, and occlusions, and it covers a range of scene types, such as indoor and outdoor environments, traffic scenes, and human activities.

The format of the data is called AER which stands for Address Event Representation [6]. According to this format, each event captured by the event camera is

denoted by a tuple consisting of four elements: (x, y, t, p). The position of the pixel where the change in brightness occurred is represented by x and y, while t denotes the exact time of the event, measured in microseconds. Finally, the polarity of the change is indicated by p, which specifies whether the brightness increased or decreased. This format of data compresses all the information required for event data into 64 bits for each event. Within the 64 bits, x and y occupy 8 bits each, timestamp occupies 32 bits, a flag takes 1 bit which represents the polarity. Figure 2 shows an exact specification of AER data format.

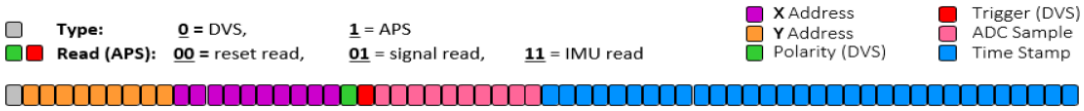


Figure 2. *Image representation of the AER data format reproduced from [6]*

2.5 Relevant Compression Methods

2.5.1 Spike Coding

The data obtained from event cameras is commonly referred to as spike data, owing to the spiking nature of the event generation process. Spike records are significantly distinct from the standard frame-based records. In order to compress this type of data, Bi et al. introduced a new technique called "spike coding" [8]. Spike coding is a cube-based coding framework that involves dividing a DVS sequence into numerous macro-cubes in the temporal domain, each having “the full spatial resolution of the pixel array” [8]. Subsequently, each macro-cube is divided into smaller cubes in a spatially distributed manner. Further details on this partitioning technique are presented in Figures 3 and 4.

The encoding process of a single spike cube involves two types of coding: spike location coding and spike polarity coding. Initially, the macro-cube is configured to the largest time-length, which is the source of the binary tree. Subsequently, “if the number of spikes in each node is less than a predefined threshold”, it is equally divided into two smaller macro-cubes using a recursive process starting from the root [8]. The 'split flag' is used to encode each node, indicating whether the current macro-cube will be split or not, as demonstrated in Figure 4.

Spike patterns are categorized into two types: “spatial-centralized and spatial-decentralized”, based on their position [8]. Moving objects tend to generate spa-

tial-centralized spikes from adjacent pixels in a DVS. On the other hand, variation in luminance tends to generate spikes globally, leading to spatially decentralized patterns. To encode these two different classes of patterns, Bi et al proposed two methods: time-prior (TP) mode and address-prior (AP) mode, respectively.

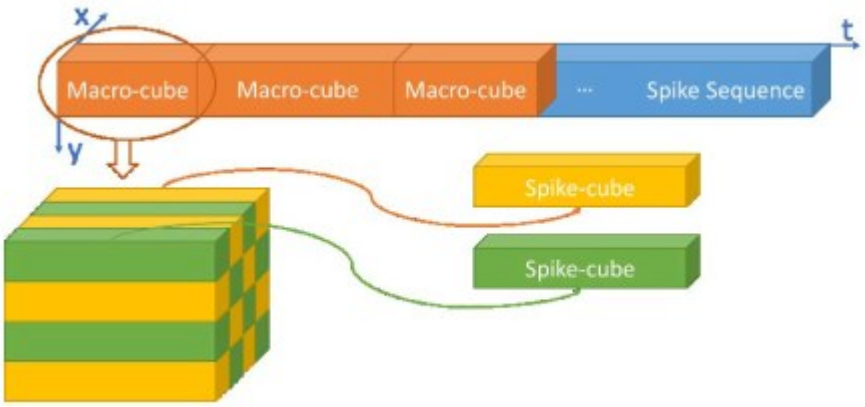


Figure 3. *Dividing macro-cubes into smaller spike-cubes from (Bi et al 2018 [8])*

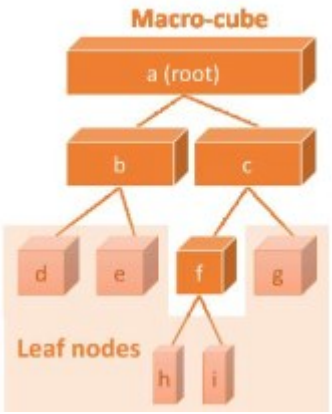


Figure 4. *Adaptive macro-cube partitioning structure from (Bi et al 2018 [8])*

The AP mode transforms the event camera data into a location histogram, which indicates the total amount of spikes at every pixel in the XY plane. After dividing this histogram into blocks, it is encoded using a context-based arithmetic coder. The context is updated adaptively with the help of adjacent blocks used as references. The current block's context is adjusted with the assistance of location histogram blocks from the preceding spike-cube. Discrete symbols are used to encode occupied blocks of the location histogram counts. The AP mode capitalizes on the smoothness of luminance changes over time, using the time intervals involving successive spikes for a given pixel to predict the occur-

rence time of the subsequent spike. Thus, “the timestamps of the spikes fired at each pixel are differentially coded sequentially” [8].

In the TP mode, first the centre point is calculated since the spike-cubes being processed in this way are spatially centralized. The centre point is calculated using formula 1 [10]:

$$x_c^*, y_c^* = \arg\min_{(x_c, y_c)} \sum_i (|x_i - x_c| + |y_i - y_c|). \quad (1)$$

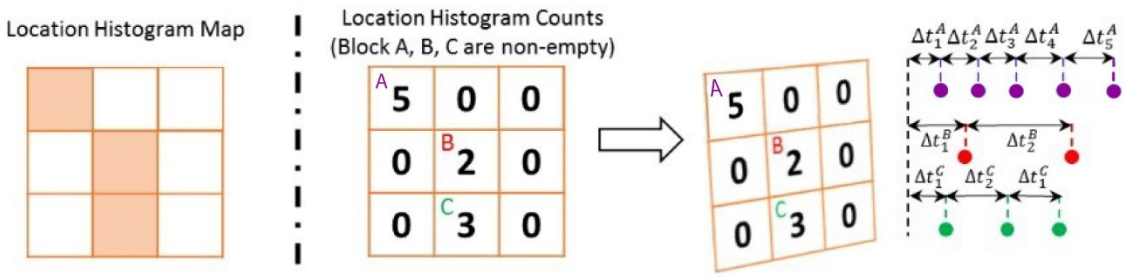


Figure 5. *Location histogram map (empty blocks are denoted by white) and the location histogram counts [8]*

In spatial terms, the spike address (x_i, y_i) represents the location of a spike in the DVS sensor. The centre point of the spike cube can be encoded either directly or differentially, by referring to the centre point of the previous spike cube. Since these spikes are centralized, their offset from the centre point forms a motion vector, which is then compressed using an entropy coder.

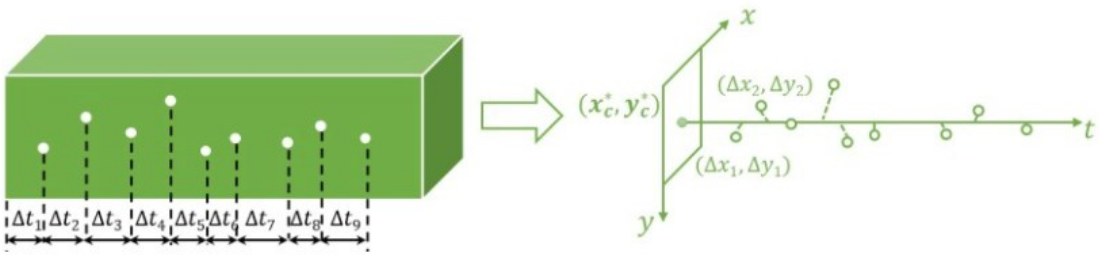


Figure 6. *Motion vector of spikes to the centre from (Bi et al 2018 [8])*

The dataset used here is PKU-DVS dataset. In summary, spike coding achieves “an average compression ratio of 19.51 against the original dataset” [8].

2.5.2 Comparison of lossless methods

Many approaches exist for compressing data. Since soon, services performed by DVS sensors will primarily be executed on the cloud/edge computing, the event data would need to transmit to the cloud. Khan et al. have implemented a multitude of existing compression technologies on event camera data for comparison [2]. They present a concise diagram on how to classify these methods and how many there are. This is presented in figure 7.

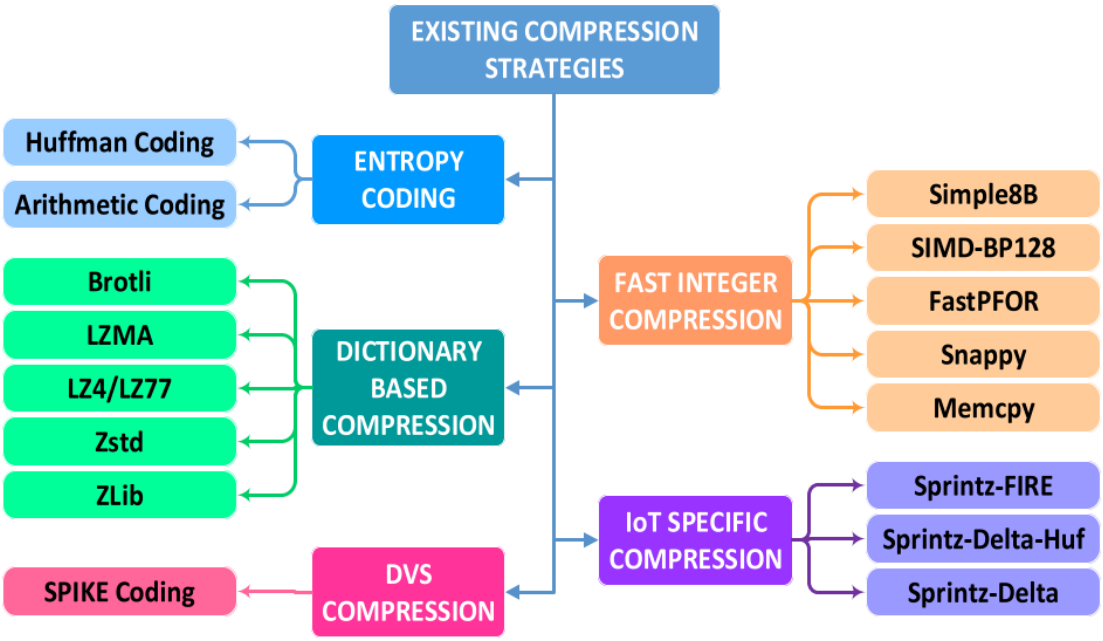


Figure 7. *Potential compression strategies for DVS data from (Khan et al 2020 [2])*

In the DVS specific compression approach, they mentioned Spike coding which is provided in more detail in section 2.4.1. Entropy encoding is a fundamental compression strategy that can be applied to all data. This is the reason that for most compression strategies, entropy encoding is used in the last compression step [2]. The concept underlying entropy encoding to utilize a reduced number of bits for data that appears more frequently, while using a “larger number of bits to represent data that occurs less frequently” [32]. This is based on the observation that, in most data sources, some values occur much more frequently than others. There are several entropy encoding techniques, including Huffman coding, arithmetic coding, and Run-Length Encoding (RLE). These methods work by creating a frequency table of the input data and using that table to assign shorter codewords to more frequent values and longer codewords to less

frequent values. The result is a compressed representation of the data that can be decompressed back to its original form with no loss of information. Huffman encoding is a type of encoding that assigns variable-length codes to symbols based on their occurrence frequency in the input data. Whereas, arithmetic coding converts a sequence of symbols into one number, which is then compressed by representing it with fewer bits [1]. Dictionary-based compression is a method of data compression that involves replacing frequently occurring data patterns or sequences with references to a dictionary of known patterns. The dictionary is usually created dynamically during the compression process and can be stored along with the compressed data to allow for later decompression.

Dictionary-based compression are methods that replace common data patterns with smaller references which in turn reduces the total size of the data. Most dictionary-based compression methods work by identifying repeated patterns in the input data and replacing them with references to a common dictionary. The LZ77 [11] compression algorithm maintains a buffer of the previous N bytes of the data being processed. This buffer is treated as a dictionary where all possible substrings of N bytes are stored as entries. There are more advanced dictionary-based compression methods, such as LZMA [12], Zstandard (Zstd) [13], Zlib [14], and Brotli [15], that use complex data structures and large dictionaries to achieve higher compression ratios. These advanced methods use multi-step encoding to achieve better compression. For example, LZMA uses multiple levels of compression, starting with dictionary-based encoding and then using an adaptive binary range coder. In contrast, Zstd employs a combination of dictionary-based coding and a hybrid entropy coder, which offers fast compression and consists of both Finite State Entropy [16] and Huffman coding. Brotli uses second order context modeling and Huffman coding along with dictionary-based encoding. Similarly, Zlib employs LZ77 and Huffman coding, allowing users to adjust the size of the sliding window buffer to optimize the balance between compression ratio and speed [1].

The authors of [17] proposed a compression algorithm for time-series data called Sprintz, which is suitable for resource-limited devices. This method achieves high compression rates with minimal memory usage (less than 1KB) and low latency. Sprintz exploits correlations between consecutive samples in

time-series data. It employs a predictor, known as Fast Integer Regression (FIRE), to encode the difference between the actual sample and the predicted sample. The prediction residuals are then processed using a bit-packing algorithm and Huffman coding. This combination of predictive coding (FIRE) and Huffman coding is called Sprintz-FIRE. To balance compression speed and ratio, Sprintz has different variations, including Sprintz-Delta, which utilizes delta coding for faster compression without Huffman coding, and Sprintz-Delta-Huf, which utilizes delta coding and Huffman encoding for better compression at the expense of lower compression speed.

Fast integer compression techniques are widely recognized for their high-speed compression and decompression capabilities, making them particularly well-suited for the encoding and decoding of vast quantities of integer arrays used in applications such as search engines and relational databases [17]. Some of these methods are SIMDBPI [18], Memcpy, FastPFOR [18] and SNAPPY [19].

Khan et al.'s experiment to compare different compression strategies made use of the DAVIS dataset [21] and PKU-DVS dataset [20]. The former was used for scenes with a mobile sensor while the latter for static sensor. In cases where the DVS is stationary, spike events are generated due to changes in lighting conditions caused by object movements in the scene. The dataset under consideration is categorized into two classifications, indoor and outdoor scenarios, with each scenario captured with varying circumstances. For example, the outdoor scenarios include day-time, night-time, and different types of movements, such as slow pedestrian movements and fast vehicle movements. Similarly, indoor scenarios feature various types of object movements, like the movement of a pendulum and human jumping movements. Table 1 reports the duration of each sequence, along with the spike event frequency, measured in Kilo-events per second "(Kev/s)", for all the indoor and outdoor scenarios. [1]

The DAVIS dataset was created to support computer vision and robotics applications that require high speed and high dynamic range in both indoor and outdoor settings. Along with providing visual output in the form of spike event streams and intensity images, the dataset also includes information on the sensor speed. The dataset covers a wide range of motion types (angular and linear) and dynamic range for both indoor and outdoor scenarios. Additionally,

the inclusion of intensity images enables the computation of scene complexity [22] for various scenes. [1]. Table 2 displays the sequences used to evaluate the compression performance, which vary in content complexity and sensor speed. The event rate is higher for scenes with greater complexity and faster sensor speeds, as reported in [22]. For example, “the Boxes sequence has an event rate of 4288.65 Kev/s”, while the Shapes sequence has a bandwidth of only “245.61 Kev/s” due to the sensor's slow speed and low content complexity. Information regarding the scene convolution and speed of the dataset can be found in [22].

Sequence		Event Rate (kev/s)	Sequence Duration (s)
Indoor	Waterdrop	3042.96	3.80
	Lighter	1329.59	2.10
	Football	1351.61	7.21
	Jump	724.09	3.28
	Game	618.42	9.57
	Pendulum	21.17	5.37
Outdoor	Pedestrians	71.66	355.21
	Daytime-Traffic1	129.70	109.84
	Daytime-Traffic2	18.32	301.60
	Night-Roadside	268.57	63.37
	Night-Traffic	350.591	15.47

Table 1. *Dataset information from PKU-DVS dataset [20]*

Sequence		Event Rate (kev/s)	Sequence Duration (s)	Scene Complexity	Speed
Indoor	Boxes	4288.65	5 (45-50)	High	High
	Poster	4021.1	5 (45-50)	High	High
	Dynamic	1077.73	20 (1-20)	Medium	Medium
	Slider	336.78	3 (1-3)	Medium	Low
	Shapes	245.61	20 (1-20)	Low	Low
Outdoor	Running3	1525.5	20 (40-60)	Medium	High
	Running2	1229.4	20 (20-40)	Medium	Medium
	Running1	713.8	20 (1-20)	Medium	Medium
	Urban	503.04	10 (1-10)	High	Low
	Walking	342.2	20 (1-20)	Medium	Low

Table 2. *Dataset information from DAVIS dataset [21]*

Many different metrics were tracked for comparison in this experiment. This thesis aims only to target the compression ratios of different compression methods. All other metrics and factors mentioned by Khan et al. [2] are out of the scope of this paper. Based on experiments conducted, it was concluded that

spike coding and dictionary-based approaches provide the best lossless compression ratios.

2.5.3 TALVEN

Khan et al. developed a compression method for event camera data called TALVEN, which stands for Time Aggregation based Lossless Video Encoding for Neuromorphic Vision sensor Data. [8] provides a mathematical analysis of the DVS spike firing mechanism, identifying two key features:

1) when there is a gradual rise or fall in luminance intensity on a particular pixel, it results in temporal connections between successive events, and

2) neighbouring pixels receive similar luminance intensities at the same time, indicating the presence of spatial redundancies.

These characteristics reveal that DVS spike sequences possess both temporal and spatial correlation, which makes them suitable for video compression. In video compression, redundancies in spatial, temporal, and numerical domains are utilized to encode data more efficiently. To take advantage of video compression techniques, the DVS event stream is transformed into a video-like format that exhibits strong temporal and spatial correlation. Figure 8 illustrates the fundamental components of this approach, “where the DVS spike event sequence is transformed into synchronous video frames” [33].

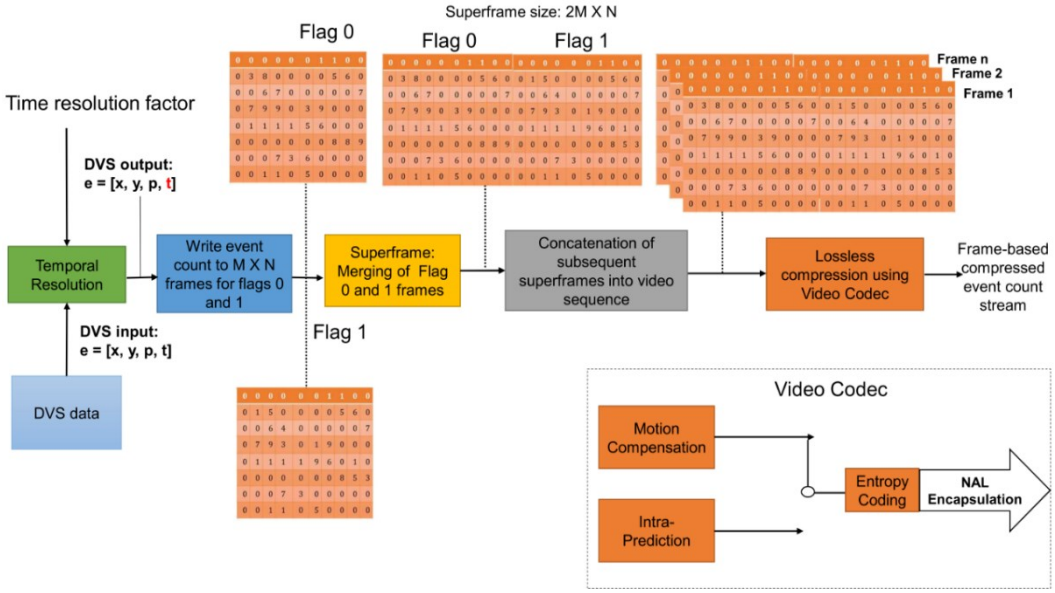


Figure 8. *Block Diagram for TALVEN from (Khan et al 2021 [23])*

To take advantage of the benefits of reducing data size by event aggregation over time, the DVS spike event stream can be transformed into a series of frames, with each frame having the full pixel array resolution. This conversion is achieved by maintaining a record of the position histogram count, which represents the amount of events at each pixel. For example, Figure 9 illustrates the location histogram count in four different scenarios with varying temporal resolution factors of 1, 2, 5, and 10 milliseconds. When the time resolution is 1 millisecond, there are 36 events with 36 bytes in total. Each pixel has one event, and every event count is shown with a byte. If the resolution is 10 milliseconds and there are 222 events, the amount of storage space needed to represent them stays the same. As a result, transforming the spike sequence into a frame that displays the event count at each pixel naturally decreases the amount of data. [23].

The event counts are calculated separately for positive and negative polarities. Doing this provides each frame to have full resolution of pixel array and increases temporal correlation between frames of the same polarity.

To enhance interframe correlation, frames of each polarity at the same time are combined to form a superframe. Although this doubles the size of the frame, it provides greater temporal correlation and includes information about polarity, as well as increases the maximum limit of event count. Despite the larger size, the benefits of superframes lead to minimal impact on potential compression gains.

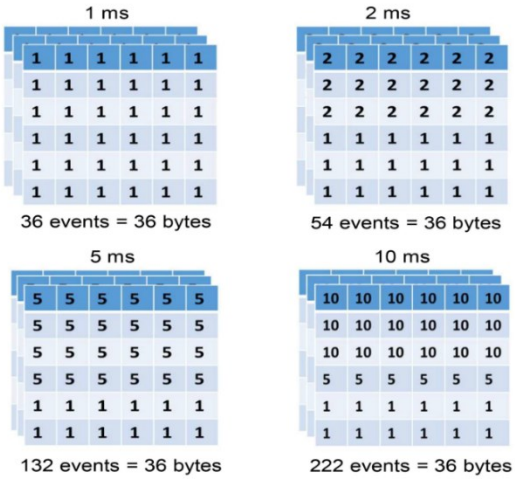


Figure 9. *Event frames showing event count at different time resolutions from (Khan et al 2021 [23])*

Once the superframe is generated, the following step is to use video encoding to represent each superframe as a video frame. The superframe contains all the pertinent spike sequence data, and interframe and intraframe coding are employed for these superframes [23]. In interframe coding, a reference frame is used to predict the current frame via clock-based motion compensated prediction (MCP) technique. The current frame is divided into blocks, and a motion vector is identified for each block in the current frame based on the address of the corresponding block in the reference frame.

The residuals of intraframe or interframe encoding are compressed using entropy encoding, followed by encapsulation using NAL (Network Abstraction Layer). NAL units are versatile as they can be mapped to various transport and network layer protocols like RTP/IP and TCP/IP. They can be packed in multiple ways, such as packet-stream format for transmission and byte-stream format for storage. NAL also aims to ensure resilience against data loss through various transmission modes like out-of-band and in-band transmission.

Algorithm 1 TALVEN

Input: Temporal resolution T_r [s]
Input: DVS spike sequence duration T_{seq} [s]
Input: Total frames to encode $T_{frames} = \frac{T_{seq}}{T_r}$

for $t = 1$ to T_{frames} **do**
  for $t = 1$ to T_r **do**
    Write event count in an $M \times N$ frame for flag 0
    Write event count in an $M \times N$ frame for flag 1
  end for
  Merge both the event frames into a superframe F for video encoding
  if $F ==$ frame type I **then**
    Apply intraframe coding
  else
    Apply interframe coding
  end if
  Apply entropy coding to the residuals of interframe or intraframe coding
  Apply NAL encapsulation to the compressed frame F
end for
Output: Compressed event frame sequence

Figure 10. *TALVEN Algorithm from (Khan et al 2021 [23])*

Their experiments utilized the DAVIS dataset. It was concluded that, compared to traditional methods, TALVEN performed better in most cases after a certain time aggregation was achieved.

2.5.4. Optical flow

Optical flow, also called optic flow or the optical flow constraint [34], is the apparent movement of objects in an image resulting from the movement of the observer in relation to the observed scene [24]. In computer vision, optical flow is generally defined as the two-dimensional vector field that describes the apparent motion of pixels between two consecutive frames in a video sequence [35]. Accurate estimation of optical flow is crucial in computer vision, as it provides information about scene motion, object motion, and camera motion. To mathematically formulate optical flow, the motion of each pixel in the image is

modelled as a function of time and space, and the parameters that best describe this motion, given the constraints of the optical flow equation, are determined. In practice, optical flow estimation is usually based on either a variational approach or a learning-based approach, both of which aim to find a mapping from image intensity at time t to image intensity at time t+1.

Event cameras have a unique application in estimating optical flow, as they can capture precise timing information for each pixel change, allowing for the direct encoding of detailed motion information in the event stream [36]. Researchers have utilized this capability to develop optical flow estimation methods using event cameras. An example of this was shown by Benosman et al., where they showed that optical flow estimation could be achieved in a linear manner by fitting a plane around each event in a local window in the spatiotemporal domain. [25].

Methods that don't require labelled data, known as self-supervised or unsupervised, have shown potential in solving complex 3D perception tasks using neural networks. Hagenaars et al. proposed a novel self-supervised approach for estimating optical flow using spiking neural networks [26]. Their approach relies solely on a representation that includes the event count per pixel and per polarity. This representation is filled with consecutive, non-overlapping partitions of the event stream called input partitions, each of which contains a fixed number of events, N.

The event stream is partitioned into small portions, each containing the same amount of events, and then formatted in sequential order for input to the network. Each partition produces an optical flow map that associates a motion vector with each input event. After processing a satisfactory quantity of events, a backward pass is conducted through a contrast maximization loss [27]. The contrast maximization approach is utilized to learn optical flows, where the exact optical flow information is contained within the spatiotemporal misalignments between the events triggered by a specific section of the moving edge. To obtain this information, motion must be compensated for by deblurring the event partition. The per-pixel optical flow "$\boldsymbol{u}(\boldsymbol{x}) = (u(\boldsymbol{x}), v(\boldsymbol{x}))^T$" is used to propagate the events to a reference time t_{ref} through formula 2 [26]:

$$\boldsymbol{x}_i' = \boldsymbol{x}_i + (t_{ref} - t_i)\boldsymbol{u}(\boldsymbol{x}_i). \quad (2)$$

To create the spiking neural network architecture called EV-FlowNet, a modified version of the network proposed in [28] is used. This architecture consists of four layer with recurrent encoders executing strided convolution, followed by ConvGRU [29], which has output feeds that double after every encoder. Following this, four decoder layers that utilize bilinear upsampling and convolution are aligned after which two residual blocks are added. Following each decoder layer, there is a connection from the corresponding encoder layer that concatenates with the activations of the previous decoder. Also, a depthwise (1 × 1) convolution is added, which produces a "lower-scale flow estimate, that is then combined with the activations of the previous decoder. The Lflow loss is applied to each intermediate optical flow estimate via upsampling. All layers utilize 3 × 3 kernels and ReLU activations, with the exception of the prediction layers that use TanH activations" [28].

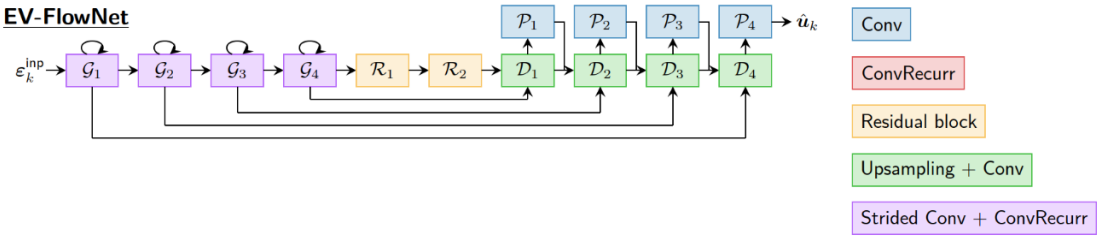


Figure 11. *Base neural network architecture used in this work from (Hagenaars et al 2021 [26])*

The researchers conducted tests to evaluate their network architectures using the MVSEC dataset [30]. The "dataset was generated for each APS frame timestamp and scaled to reflect the displacement for either one (dt = 1) or four (dt = 4) APS frames. Optical flow predictions were produced at each frame timestamp by utilizing all events in the time window as input for dt = 1, or 25% of the window events at a time for dt = 4", owing to the greater displacements [26]. The results of the experiment are shown in Table 3. Since EV-FlowNet has the smallest AEE, it is selected as the best network and is subsequently utilized in this study.

dt = 1	outdoor_day1		indoor_flying1		indoor_flying2		indoor_flying3	
	AEE	%Outlier	AEE	%Outlier	AEE	%Outlier	AEE	%Outlier
EV-FlowNet* [54]	0.49	0.20	1.03	2.20	1.72	15.10	1.53	11.90
EV-FlowNet* [56]	**0.32**	**0.00**	**0.58**	**0.00**	**1.02**	**4.00**	**0.87**	**3.00**
Hybrid-EV-FlowNet* [26]	0.49	-	0.84	-	1.28	-	1.11	-
EV-FlowNet	0.47	0.25	0.60	0.51	1.17	8.06	0.93	5.64
RNN-EV-FlowNet	0.56	1.09	0.62	0.97	1.20	8.82	0.93	5.51
Leaky-EV-FlowNet	0.53	0.28	0.71	0.60	1.43	11.37	1.14	8.12
LIF-EV-FlowNet	0.53	0.33	0.71	1.41	1.44	12.75	1.16	9.11
ALIF-EV-FlowNet	0.57	0.42	1.00	2.46	1.78	17.69	1.55	15.24
PLIF-EV-FlowNet	0.60	0.52	0.75	0.85	1.52	13.38	1.23	9.48
XLIF-EV-FlowNet	0.45	0.16	0.73	0.92	1.45	12.18	1.17	8.35
FireNet	0.55	0.35	0.89	1.93	1.62	14.65	1.35	10.64
RNN-FireNet	0.62	0.52	0.96	2.60	1.77	17.55	1.48	13.60
Leaky-FireNet	0.52	0.41	0.90	2.66	1.67	16.09	1.43	13.16
LIF-FireNet	0.57	0.40	0.98	2.48	1.77	16.40	1.50	12.81
ALIF-FireNet	0.62	0.45	1.04	3.02	1.85	18.88	1.58	15.00
PLIF-FireNet	0.56	0.38	0.90	1.93	1.67	14.47	1.41	11.17
XLIF-FireNet	0.54	0.34	0.98	2.75	1.82	18.19	1.54	14.57
dt = 4								
EV-FlowNet* [54]	1.23	**7.30**	2.25	24.70	4.05	45.30	3.45	39.70
EV-FlowNet* [56]	1.30	9.70	2.18	24.20	3.85	46.80	3.18	47.80
Hybrid-EV-FlowNet* [26]	**1.09**	-	2.24	-	**3.83**	-	3.18	-
EV-FlowNet	1.69	12.50	**2.16**	**21.51**	3.90	**40.72**	**3.00**	**29.60**
RNN-EV-FlowNet	1.91	16.39	2.23	22.10	4.01	41.74	3.07	30.87
Leaky-EV-FlowNet	1.99	17.86	2.59	30.71	4.94	54.74	3.84	42.33
LIF-EV-FlowNet	2.02	18.91	2.63	29.55	4.93	51.10	3.88	41.49
ALIF-EV-FlowNet	2.13	20.96	3.81	50.36	6.40	66.03	5.53	61.07
PLIF-EV-FlowNet	2.24	23.76	2.80	34.34	5.21	52.98	4.12	45.31
XLIF-EV-FlowNet	1.67	12.69	2.72	31.69	4.93	51.36	3.91	42.52
FireNet	2.04	20.93	3.35	42.50	5.71	61.03	4.68	53.42
RNN-FireNet	2.35	24.31	3.64	46.54	6.33	63.89	5.20	56.60
Leaky-FireNet	1.96	18.26	3.42	42.03	5.92	58.80	4.98	52.57
LIF-FireNet	2.12	21.00	3.72	48.27	6.27	64.16	5.23	58.43
ALIF-FireNet	2.36	25.82	3.94	52.35	6.65	67.61	5.60	61.93
PLIF-FireNet	2.11	20.64	3.44	44.02	5.94	64.02	4.98	57.53
XLIF-FireNet	2.07	18.83	3.73	47.89	6.51	67.25	5.43	60.59

Table 3. *Quantitative evaluation on MVSEC, table reproduced from Zhu et al. 2021 [30]. Average Endpoint Error (lower is better) in pixels and the percentage of outliers. Best in bold, runner up underlined.*

3. PROPOSED COMPRESSION STRATEGIES

3.1 Compressing individual event streams using existing methods

The first approach utilized in this study involves the isolation and compression of individual components of the dataset, namely x, y, timestamp, and polarity, separately. The primary objective of this method was to attain lossless compression, which requires the same resolution after decompression. For DAVIS sensors, the resolution required is 240x180 pixels, as stated in section 2.4. The dataset used in this study provides AEDAT files that can be loaded and read into MATLAB for data extraction. In this baseline compression approach, each data stream was separately compressed using established compression technologies, namely PPmD, Bzip2, and LZMA. These algorithms were selected based on the findings in [2], as discussed in greater detail in section 2.4.2, which elaborates on the rationale behind their selection as the most effective options for lossless compression.

LZMA is a lossless compression algorithm that combines LZ77 and LZ78 algorithms with a Markov chain model to achieve high compression ratios, as explained in [2]. The algorithm first creates a dictionary of substrings present in the input data, which is used to identify and represent repeated substrings with a reference to their previous occurrence, along with the length of the substring, using the LZ77 algorithm [16]. The LZMA algorithm enhances this process by utilizing variable-length encoding to represent the reference and length information, enabling more efficient compression of repeated substrings. The remaining uncompressed data is encoded using the LZ78 algorithm, which builds a dictionary of previously encountered sequences and represents new sequences with references to dictionary entries. Additionally, the LZMA algorithm uses a Markov chain model to predict the next character in the input data based on previous characters, and encodes it using a variable-length code, reducing the size of compressed data by exploiting the statistical properties of input data. The algorithm is highly configurable, allowing customization of parameters such

as dictionary size, match finder algorithm, and compression level. 7-Zip, WinZip, and WinRAR frequently use LZMA algorithm due to its high compression ratio and relatively fast decompression speed. However, the algorithm's high computational demands make it less suitable for low-power or resource-constrained devices.

PPMD (Prediction by Partial Matching) is a lossless data compression algorithm that is based on statistical modelling and prediction of the input data. The algorithm uses a combination of context modelling and adaptive probability estimation to achieve high compression ratios. The PPMD algorithm works by building a model of the input data using a set of previous symbols. The model is based on the probability distribution of the symbols that occur in the input data given the context of the previous symbols. The context can range from a single symbol to several symbols, depending on the size of the context window used by the algorithm [31]. The algorithm utilizes a range of different techniques to optimize the model building and probability estimation process. For example, it employs partial matching to find the longest context that matches the current input data and then uses this context to predict the next symbol. Additionally, the algorithm uses adaptive probability estimation, meaning that it adjusts the probability estimates for each symbol based on the actual occurrences of the symbols in the input data [31]. One of the advantages of the PPMD algorithm is its ability to adapt to the statistical properties of the input data, making it particularly effective at compressing data with complex patterns and structures. However, the algorithm is computationally intensive, making it less suitable for use in low-power or resource-constrained devices. Additionally, the compressed data produced by the algorithm may be larger than that produced by other algorithms when applied to certain types of data with low entropy.

BZip2 is a lossless data compression algorithm that employs statistical modelling and Burrows-Wheeler Transform (BWT) to achieve high compression ratios. The BZip2 algorithm utilizes a multi-step approach that involves applying the BWT to the input data to group similar characters together, followed using Move-to-Front (MTF) transform, Run-Length Encoding (RLE), Huffman coding, and selective BWT of smaller data blocks to compress the transformed data. The MTF transform encodes symbols based on their position in a symbol table,

while the RLE step compresses the data by encoding repeated sequences as symbol-length pairs. The Huffman coding step further compresses the RLE data by assigning shorter codes to frequently occurring symbols and longer codes to less frequent ones. The selective BWT step divides the transformed data into smaller blocks and applies the BWT to each block separately, enabling the algorithm to handle larger data sets and improving the efficiency of the Huffman coding step. One of the notable strengths of BZip2 is its ability to effectively handle diverse data types and structures through the use of a large symbol table for encoding the input data.

In this study, we utilized compression algorithms on each data stream in our dataset, specifically on the PKU dataset's pendulum data. Figure 12 displays a subset of the original pendulum dataset values, providing insight into the data's distribution across all data components within the AER data format. The dataset consists of 113,682 events in total. Additionally, Figure 13 illustrates the frequency of each individual data stream. The non-linear pattern of the data is evident, as the first 500 events exhibit close proximity in coordinates, while the rest of the dataset may not follow this same pattern.

In order to compress the datasets, the individual data streams were initially saved to disk as binary files, utilizing 16-bit integers for x and y coordinates, unsigned bits for polarity, and 32-bit integers for timestamps. Subsequently, the compression techniques were implemented on these files.

After compression was achieved, the compressed files were decompressed and rechecked for any loss. Since this thesis only focuses on lossless compression, it was vital to perform decompression and equality check after the algorithms were applied. Following are the results from this experiment on the PKU dataset.

It is evident that PPmD outperforms the other two selected compression algorithms in the present study. This can be attributed to the fact that PPmD is a dictionary-based compression algorithm and the dataset used here exhibits a high degree of spatial correlation. Since events predominantly occur at the edge of the frame due to the moving scene, there are many points in the same

neighbourhood, and PPmD can effectively predict probabilities based on context to compress the dataset. While PPmD generally yields better compression

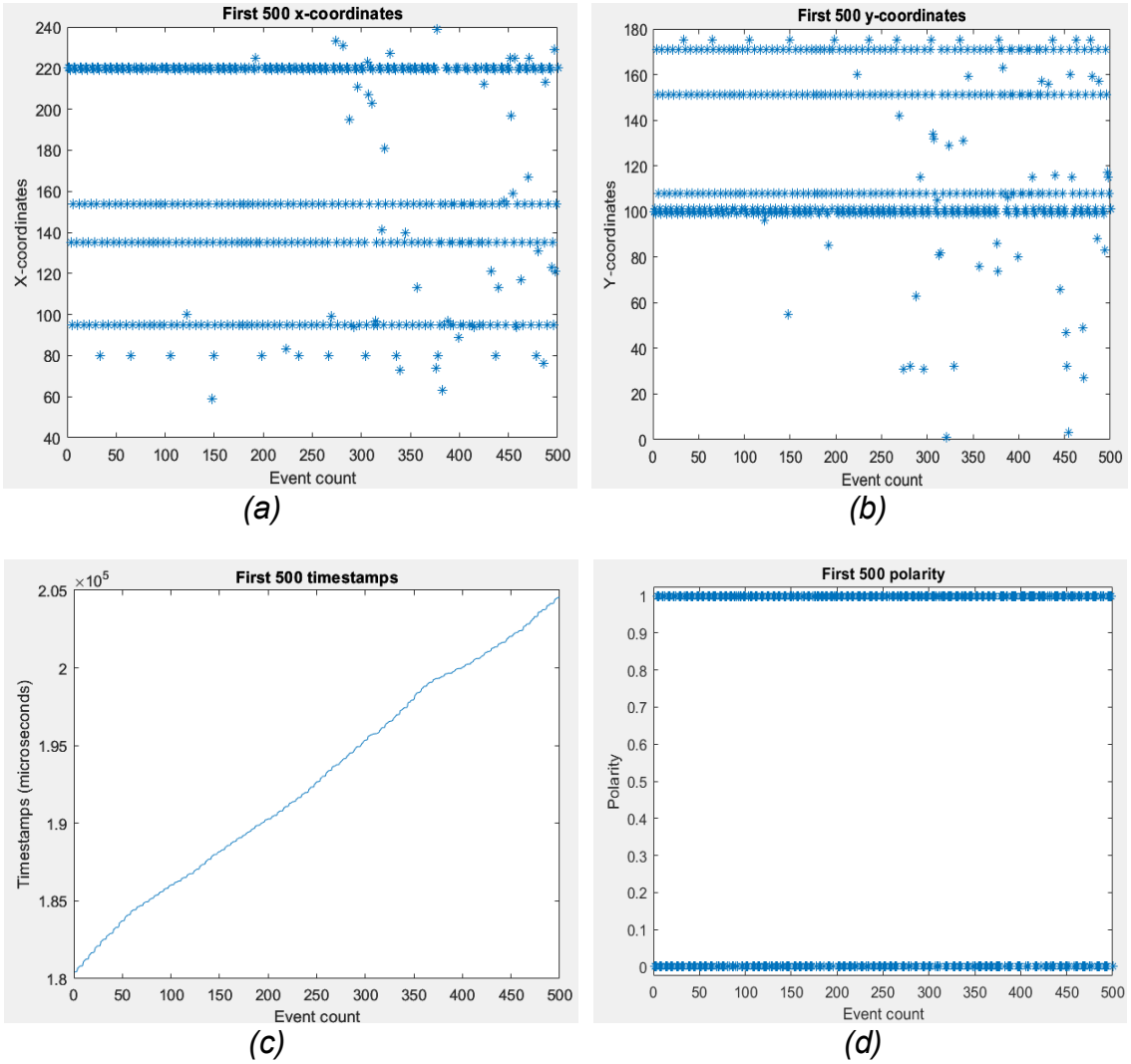
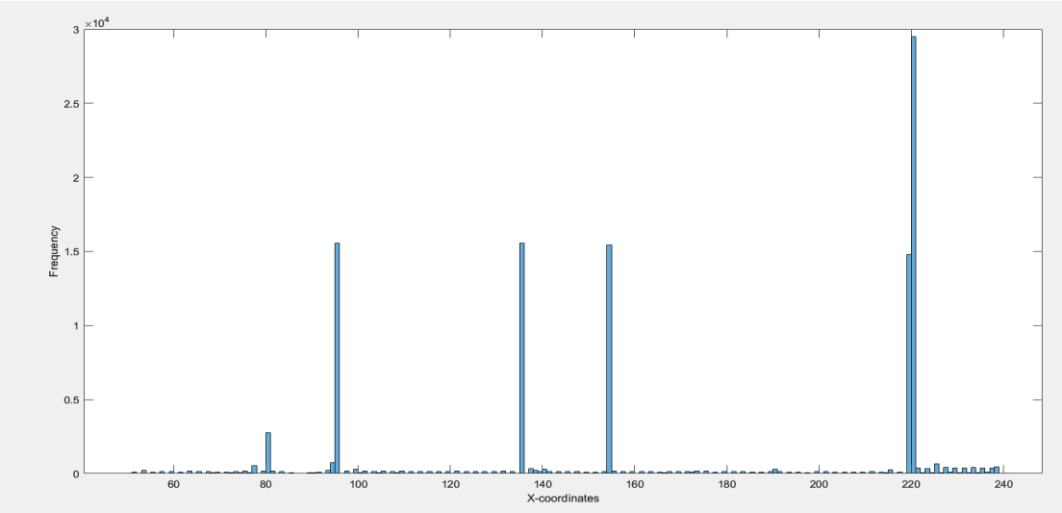
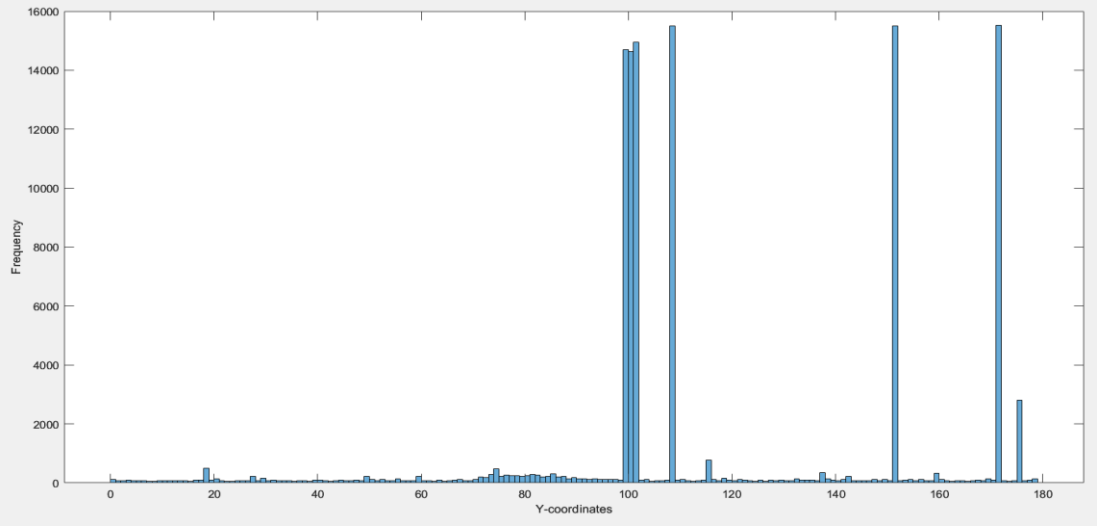


Figure 12. *Original dataset value. The first 500 x-coordinates (a), y-coordinates (b), timestamps (c) and polarity (d) from Pendulum, PKU dataset*

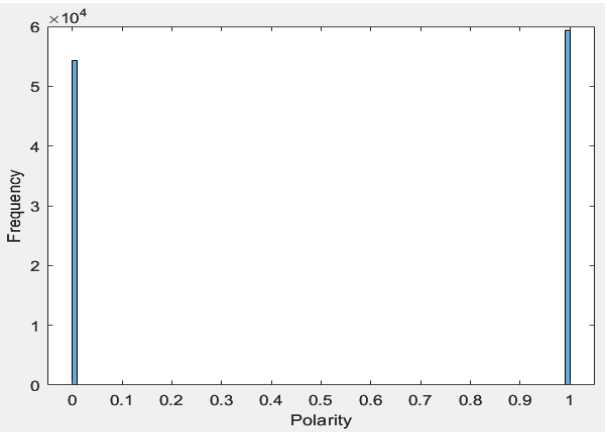
results for x-coordinates, Bzip2 performs slightly better in certain cases, which can be attributed to the repetitive pattern of occurring coordinates. The intersection dataset demonstrates the highest repetition of x-coordinates, and Bzip2 achieves nearly five times better bits per event ratio for this dataset. However, since Bzip2 is not optimal for compressing binary files, which is the file format in this study, it is not the best performing method. In contrast, LZMA performs slightly better for the pedestrian dataset due to its exceptional performance in compressing large data sets.



(a)



(b)



(c)

Figure 13. *Histograms showing frequencies of x-coordinate (a), y-coordinate (b) and polarity (c) values for Pendulum, PKU dataset*

Sequence	Number of events	Original Size (bits)	LZMA	Bits per event LZMA	PPmD	Bits per event PPmd	BZip2	Bits per event BZip2
Waterdrop	11,563,243	185,011,888	1,960,584	0.1695	**1,086,392**	**0.0939**	1,378,720	0.1192
Lighter	2,792,139	44,674,224	3,549,360	1.2712	3,912,296	1.4011	**3,195,792**	**1.1445**
Football	9,745,101	155,921,616	11,153,432	1.145	**10,709,328**	**1.0989**	11,200,296	1.1493
Jump	2,375,022	4,750,044	6,372,880	2.6833	**6,320,872**	**2.6614**	6,385,416	2.6886
Game	5,918,277	94,692,432	33,047,968	5.5841	**32,319,856**	**5.4610**	33,367,120	5.6380
Pendulum	113,682	1,818,912	308,488	2.7136	**258,720**	**2.2758**	308,416	2.7130
Pedestrians	25,455,052	407,280,832	**24,293,528**	**0.9543**	29,554,928	1.1611	24,956,328	0.9804
Daytime-traffic1	14,246,079	227,937,264	21,297,472	1.4949	23,475,320	1.6478	**20,082,208**	**1.4096**
Daytime-traffic2	5,525,453	88,407,248	11,525,880	2.0859	**10,601,368**	**1.9186**	11,977,928	2.1677
Night-Roadside	17,018,997	272,303,952	67,174,096	3.9470	**64,528,712**	**3.7915**	66,750,576	3.9221
Night-Traffic	105,631,856	690,109,696	582,277,496	5.5123	**540,553,872**	**5.1173**	583,263,904	5.5216
Intersection	2,016,922	32,270,752	386,096	0.191	1,131,096	0.5608	**208,248**	**0.1032**
Fluorescent	12,839,145	205,426,320	8,546,056	0.6656	**6,310,176**	**0.4914**	7,871,488	0.6131

Table 4. *Encoding x-coordinates using 7zip, best in bold*

Sequence	Number of events	Original Size (bits)	LZMA	Bits per event LZMA	PPmD	Bits per event PPmD	BZip2	Bits per event BZip2
Waterdrop	11,563,243	185,011,888	1,960,584	0.1696	**1,086,392**	**0.0940**	1,378,720	0.1192
Lighter	2,792,139	44,674,224	2,303,184	0.8248	2,676,192	0.9584	**1,945,792**	**0.6968**
Football	9,745,101	155,921,616	12,208,904	1.2528	**10,818,136**	**1.1101**	11,381,712	1.1679
Jump	2,375,022	38,000,352	5,616,440	2.3648	**5,049,744**	**2.1262**	5,052,064	2.1272
Game	5,918,277	94,692,432	18,456,872	3.1186	**157,46,824**	**2.6607**	17,981,168	3.0382
Pendulum	113,682	1,818,912	304,776	2.6810	**,240,968**	**2.1197**	280,184	2.4646
Pedestrians	25,455,052	407,280,832	21,257,600	0.8351	22,898,544	0.8996	**18,217,960**	**0.7157**
Daytime-traffic1	14,246,079	227,937,264	18,333,920	1.2869	19,134,288	1.3431	**15,834,384**	**1.1115**
Daytime-traffic2	5,525,453	88,407,248	10,338,800	1.8711	**7,757,568**	**1.4040**	9,322,456	1.6872
Night-Roadside	17,018,997	272,303,952	37,938,376	2.2292	**32,686,624**	**1.9206**	33,396,912	1.9623
Night-Traffic	105,631,856	1690,109,696	331,366,016	3.1370	**272,820,512**	**2.5827**	311,865,496	2.9524
Intersection	2,016,922	32,270,752	389,280	0.1930	1,135,488	0.5630	**215,352**	**0.1068**
Fluorescent	12,839,145	205,426,320	7,720,080	0.6013	**4,919,704**	**0.3832**	6,411,128	0.4993

Table 5. *Encoding y-coordinates using 7zip, best in bold*

Similar to the compression of x-coordinates, the results indicate that PPmD is the most effective algorithm for compressing y-coordinates. PPmD is particularly well-suited for structured data formats as it can leverage the well-defined structure to identify and exploit patterns in the input stream. It is particularly effective for data that contains a high level of redundancy, as it can accurately predict the probability of different symbols in the input stream based on the context of previous symbols.

Sequence	Number of events	Original Size (bits)	LZMA	Bits per event LZMA	PPmD	Bits per event PPmD	BZip2	Bits per event BZip2
Waterdrop	11,563,243	370,023,776	3,143,672	0.2719	15,616,184	1.3505	**1,662,576**	**0.1438**
Lighter	2,792,139	89,348,448	1,580,888	0.5662	4,085,864	1.4633	**1,126,456**	**0.4034**
Football	9,745,101	311,843,232	5,931,640	0.6087	14,751,456	1.5137	**4,050,928**	**0.4157**
Jump	2,375,022	76,000,704	3,399,768	1.4315	4,609,520	1.9408	**2,756,648**	**1.1607**
Game	5,918,277	189,384,864	11,423,744	1.9302	12,678,920	2.1423	**9,763,544**	**1.6497**
Pendulum	113,682	3,637,824	742,760	6.5337	741,368	6.5214	**736,360**	**6.4774**
Pedestrians	25,455,052	814,561,664	29,974,776	0.8326	53,625,608	1.7890	**28,956,936**	**0.9660**
Daytime-traffic1	14,246,079	455,874,528	22,432,248	1.5746	33,958,104	2.3837	**21,591,272**	**1.5155**
Daytime-traffic2	5,525,453	176,814,496	22,866,560	1.6051	22,548,376	1.5828	**22,074,680**	**1.5495**
Night-Roadside	17,018,997	544,607,904	36,848,584	2.1651	41,475,424	2.4370	**31,963,848**	**1.8781**
Night-Traffic	105,631,856	3380,219,392	262,916,976	2.4890	263,184,888	2.4915	**226,898,000**	**2.1480**
Intersection	2,016,922	64,541,504	221,528	0.1098	2,760,056	1.3684	**132,680**	**0.0658**
Fluorescent	12,839,145	410,852,640	2,679,944	0.2087	17,942,456	1.3975	**1,747,416**	**0.1361**

Table 6. *Encoding difference timestamps using 7zip, best in bold*

In the compression of timestamps, a method was employed to reduce the large size of integers by creating difference times. The timestamps are stored as integers with high precision, occupying several bits when encoded. Since timestamps are stored as integers and are accurate to the millisecond, these integer numbers are large. By subtracting each timestamp from the previous one, a sequence of smaller integers is generated. This reduces the overall size of integers and saves bits during compression. Additionally, the repeated occurrence of many numbers can be leveraged to further improve compression. Figure 14 shows the effectiveness of this approach.

The analysis reveals that Bzip2 outperforms other compression algorithms in all scenarios when compressing difference timestamps. Bzip2 is known for its ability to compress data that contains random or unpredictable patterns, as it can efficiently identify and exploit patterns in the input stream. As the difference timestamp data lacks a well-defined structure and contains rather random values, Bzip2 offers a better bits per event ratio for compressing this data.

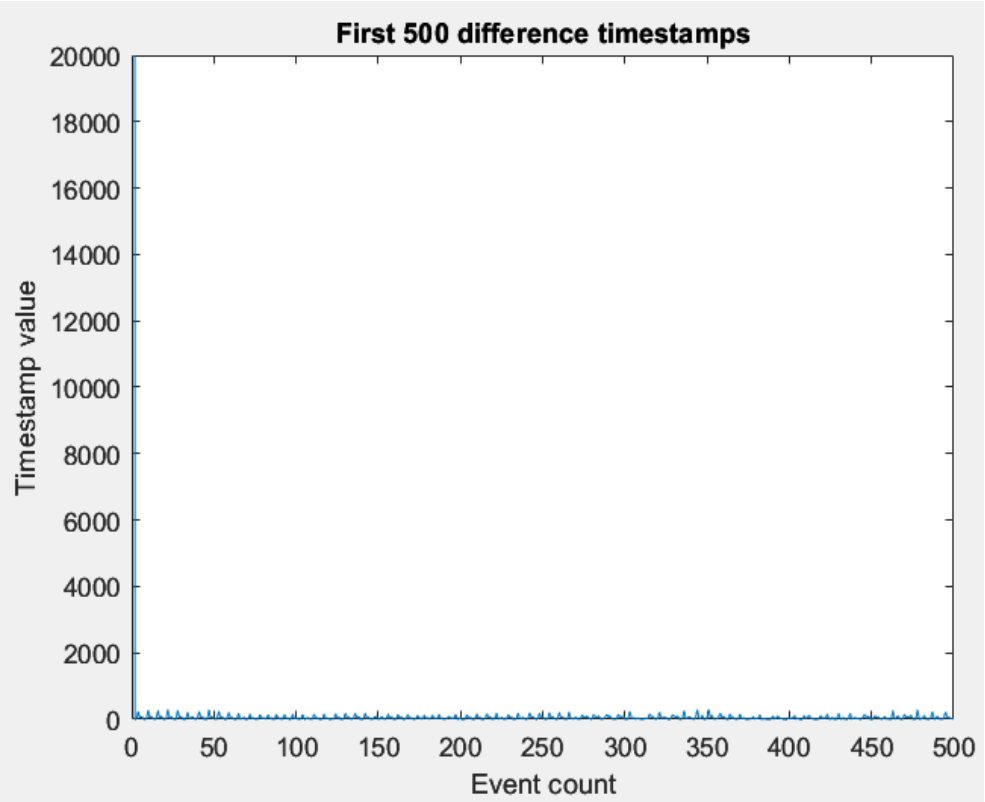


Figure 14. *Difference timestamp data created for Pendulum, PKU dataset. Original timestamps in figure 12(c)*

Sequence	Number of events	Original Size (bits)	LZMA	Bits per event LZMA	PPmD	Bits per event PPmD	BZip2	Bits per event BZip2
Waterdrop	11,563,243	11,563,248	8,02,360	0.0694	**616,808**	**0.0533**	590,760	0.0511
Lighter	2,792,139	2,792,144	4,17,392	0.1495	**380,840**	**0.1364**	423,248	0.1516
Football	9,745,101	9,745,104	29,94,392	0.3073	**2,739,432**	**0.2811**	3,084,584	0.3165
Jump	2,375,022	2,375,024	8,77,512	0.3695	**855,592**	**0.3602**	949,848	0.3999
Game	5,918,277	5,918,280	31,00,224	0.5238	**3,060,784**	**0.5172**	3,360,168	0.5678
Pendulum	113,682	113,688	84,800	0.7459	**82,888**	**0.7291**	91,288	0.8030
Pedestrians	25,455,052	25,455,056	5,801,424	0.1935	**5,583,144**	**0.1863**	5,832,856	0.1946
Daytime-traffic1	14,246,079	14,246,080	4,158,320	0.2919	**4,077,064**	**0.2862**	4,281,416	0.3005
Daytime-traffic2	5,525,453	5,525,456	3,154,408	0.2214	**3,057,096**	**0.2146**	3,387,840	0.2378
Night-Roadside	17,018,997	17,019,000	8,821,408	0.5183	**8,610,016**	**0.5059**	9,395,560	0.5521
Night-Traffic	105,631,856	105,631,856	44,557,048	0.4218	**42,841,160**	**0.4056**	46,278,400	0.4381
Intersection	2,016,922	2,016,928	,111,944	0.0555	81,968	0.0406	**78,048**	**0.0387**
Fluorescent	12,839,145	12,839,152	2,042,840	0.1591	**1,916,576**	**0.1493**	2,082,288	0.1622

Table 7. *Encoding polarity using 7zip, best in bold*

The polarity information, represented by a single bit indicating 'on' or 'off', is repeated frequently in each dataset. The repetitive pattern of this binary data makes it ideal for compression. As predicted, PPmD outperforms the other algo-

rithms in compressing this type of data, as it utilizes the context of the current symbol to predict the next symbol efficiently. After compressing each stream of data individually, the most effective compression method for each stream was selected and combined to compute the final compression ratio. The compression ratio was calculated by dividing the number of bits required to store an event in the original AER format (64) by the number of bits per event required after compression. The results are summarized in Table 8.

A similar study was performed on the DAVIS dataset since the motion patterns comprised in each dataset is unique. The results of this experiment are demonstrated in table 9.

It can be noted that this particular method performs a lot better on the PKU dataset and is comparable to the results in [23]. The DAVIS dataset does not exhibit high compression ratios like PKU dataset which can be attributed to the more complex movement patterns and motion of the DVS.

Sequence	Bits per event for compressing rows using PPmD	Bits per event for compressing columns using PPmD	Bits per event for compressing XY (rows and columns)	Bits per event for compressing timestamps using Bzip2	Bits per event for compressing polarity using PPmD	Total bits per event	Compression Ratio (compared to 64 bits)
Waterdrop	0.0939	0.0940	0.1879	0.1438	0.0533	0.3812	167.8908
Lighter	1.4011	1.4012	2.8023	0.4034	0.1364	3.3421	19.1496
Football	1.0989	1.0989	2.1978	0.4157	0.2811	2.8946	22.1101
Jump	2.6614	2.6614	5.3228	1.1607	0.3602	6.9298	9.2354
Game	5.4610	5.4610	10.922	1.6497	0.5172	13.0889	4.8896
Pendulum	2.2758	2.2758	4.5516	6.4774	0.7291	11.7801	5.4329
Pedestrians	1.1611	1.1611	2.3222	0.9660	0.1863	4.0173	15.9310
Daytime-traffic1	1.6478	1.6478	3.2956	1.5155	0.2862	5.0973	12.5556
Daytime-traffic2	1.9186	0.7442	2.6628	1.5495	0.2146	4.4269	14.4571
Night-Roadside	3.7915	3.7916	7.943	1.8781	0.5059	10.327	6.1973
Night-Traffic	5.1173	5.1173	10.2346	2.1480	0.4056	12.7882	5.0046
Intersection	0.5608	0.5608	1.1216	0.0658	0.0406	1.228	52.1172
Fluorescent	0.4914	0.4915	0.983	0.1361	0.1493	1.2684	50.4573
Total Average							29.6483

Table 8. *Baseline compression on PKU dataset*

Sequence	Bits per event for compressing rows using PPmD	Bits per event for compressing columns using PPmD	Bits per event for compressing XY (rows and columns)	Bits per event for compressing timestamps using LZMA	Bits per event for compressing polarity using LZMA	Total bits per event	Compression Ratio
Slider	2.8648	5.1358	8.0006	3.0909	0.8466	11.9381	5.3609
Shapes	4.4283	5.4250	9.8533	4.1815	0.9986	15.0334	4.2571
Dynamic	5.1260	4.9716	10.0977	3.3646	0.9788	14.4411	4.4317
Boxes	3.3027	4.5166	7.8193	3.1945	0.9821	11.9959	5.3351
Poster	3.8589	4.1764	8.0353	1.9653	0.9825	10.9831	5.8271
Running	6.4119	4.6035	11.0154	2.4872	0.9837	14.4863	4.4179
Urban	5.6483	4.8000	10.4483	3.6569	0.9576	15.0628	4.2488
Walking	6.5621	4.9355	11.4976	3.6897	0.9647	15.1520	4.2238
Slider Depth	5.1909	5.1934	10.3843	3.9562	0.8586	15.1991	4.2107
Shapes 6dof	5.8376	4.3781	10.2157	4.1511	0.9980	15.3648	4.1653
Dynamic 6dof	5.1012	4.9622	10.0634	2.8636	0.9794	13.9064	4.6021
Calibration	6.3261	4.8964	11.2225	3.9888	0.9690	16.1803	3.9554
Office Zigzag	5.8369	5.1099	10.9468	2.9964	0.9740	14.9172	4.2903
Total Average							4.5635

Table 9. *Baseline compression on DAVIS dataset*

3.2 Creating and encoding location images using x and y coordinates

In the second proposed method, the encoding of event locations (x and y) is achieved by isolating these components and encoding them as a location image. Meanwhile, timestamps and polarity are encoded using the previous method with Bzip2 and PPmD, respectively. To ensure genuine compression without distorting the event location image, a specific number of bits, referred to as ‘nBits,’ must be set. The size of nBits determines the number of events that will be encoded in each image and, consequently, the size of the block in which events are split. Choosing the appropriate value for nBits is crucial as a larger value may result in an unclear image capturing too many events per frame, while a smaller value may produce an image with fewer events than required. Figure 15 displays the resulting images and their differences for various nBits values. After conducting multiple experiments, we determined that 11 bits worked well for all datasets. Additionally, the resolution of the dataset must be known. For our case, the DAVIS240b dataset has a resolution of 240x180 pixels.

To create event location images using this method, the entire set of x and y coordinates, the total number of events, and the number of events per image (M) are required. The process involves counting the first M events from the beginning (i.e., 0) and creating a block of M events each for x and y. All linear indices in these blocks are found using the first 50 values for x and y, and all these indices are assigned a value of 1 in a temporary array. This array demonstrates the locations where an event occurred, and a single iteration of this process is demonstrated in Figure 16.

Upon identification of event locations, the subsequent step involves encoding them. The summation of events in the location image is determined, and if the total number of events falls below 2nbits and above 2nbits - M, a frame number is allocated to this specific image and it is encoded. To carry out the encoding process, the length of the block is computed, and a bitstream of zeros is generated with a length equal to (nBits * length of block). Afterward, all values in the image that indicate an event occurrence are identified, and a new temporary image is created. Subsequently, bit values from this image are incorporated into the bitstream, nBits at a time, until all image locations have been encoded. The resulting location image is written to disk, with PNG and LOCO serving as the encoding mechanisms.

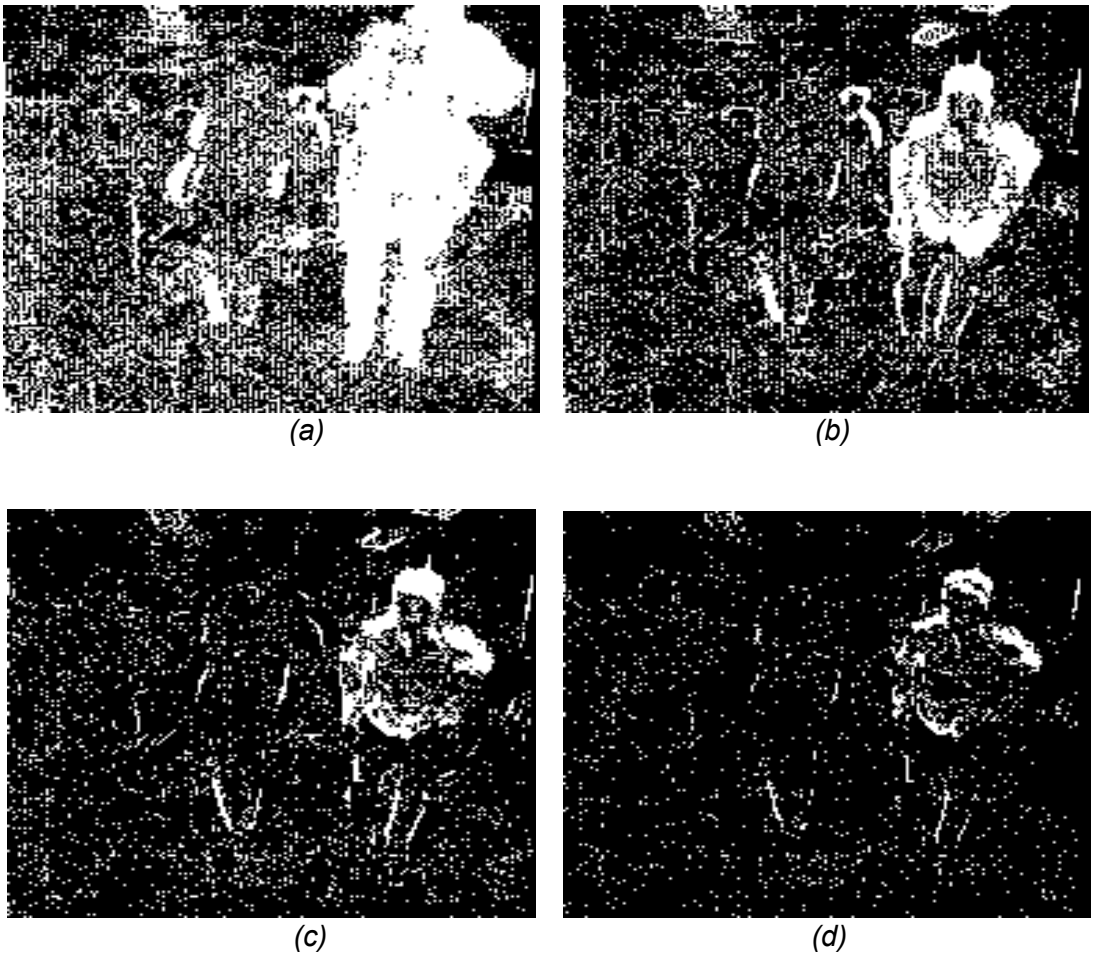


Figure 15. *First image frame generated by collecting x and y location co-ordinates. Block size is 2^n bits where n = 14 bits (a), 13 bits (b), 12 bits (c) and 11 bits (d) for Jump, PKU data set*

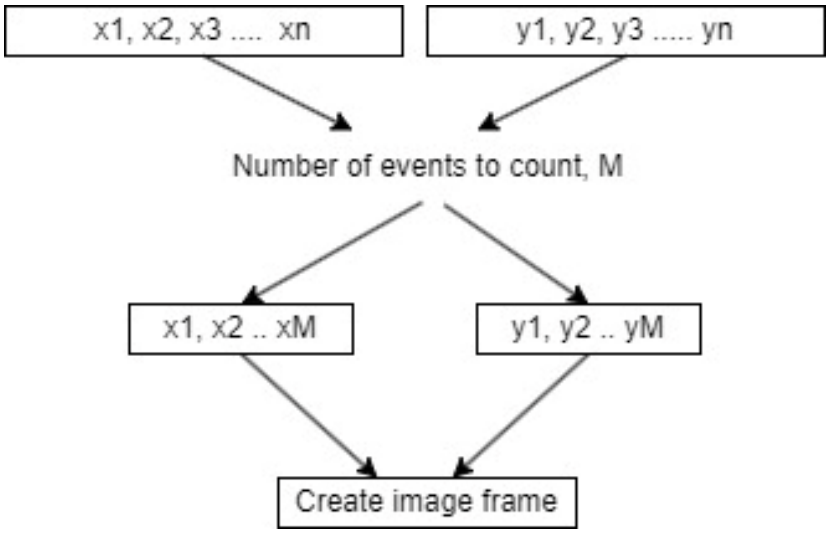


Figure 16. *Creating location images using a blocking mechanism*

PNG format is a lossless image compression method that ensures that the compressed image can be reconstructed perfectly without any loss of information. PNG utilizes several compression techniques, including deflate compression and filter algorithms, to compress the image data. To reduce the amount of data that needs to be compressed, PNG first applies one of several filter algorithms to the image data. These filter algorithms analyse each row of pixel data and try to remove any redundant information by predicting the values of each pixel based on the values of surrounding pixels. The filtered image data is then compressed using the deflate compression algorithm, which identifies repeated patterns in the data and replaces them with shorter codes using lossless compression. Additionally, PNG employs Huffman coding to further compress the data by assigning shorter codes to frequently occurring patterns in the data and longer codes to less frequent patterns.

LOCO method utilizes a combination of predictive coding, context modelling, and entropy encoding to compress images. Predictive coding is used to encode the image data by predicting the value of each pixel based on the values of neighbouring pixels and then encoding the difference between the predicted value and the actual value of the pixel. This method reduces the amount of data that needs to be compressed. After predictive coding is applied, context modelling is used to analyse the data and predict the most likely value for each pixel based on its surrounding pixels. This reduces the number of bits required to represent each pixel in the compressed data. Finally, the compressed data is encoded using entropy coding techniques such as arithmetic coding or Golomb-Rice coding. Entropy coding assigns shorter codes to more frequently occurring data patterns, further reducing the overall size of the compressed data.

2^nbits	Total bits needed to compress an image frame using PNG	Total bits needed to compress an image frame using Loco
11	12.4274	11.9332
12	12.7117	12.4916
13	13.3379	13.2413
14	14.1141	14.0844

Table 10. *Number of bits needed to encode location image for 'Jump' from PKU*

Table 10 demonstrates that LOCO outperforms PNG in compressing images from the PKU dataset. Figure 15 confirms that using a larger value for nBits re-

sults in images with more noise, while an appropriate value for nBits retains crucial details of the events. However, nBits values lower than 11 may cause significant loss of information. Notably, LOCO reduces the number of bits required for encoding x and y coordinates from the original 15 to slightly over 3, which is impressive. The following pseudocode summarizes this method, and the outcomes are presented in Table 11.

Algorithm: Encoding event locations

```
Input : x
Input : y
Input : Number of events
Input : Number of events to count, M
framenumber = 0
for i = 1 to number of events do
   currentXs=x(0:M)
   currentYs=y(0:M)
   temporaryImage=zeros(180,240)
   ind=linear indices of currentXs and currentYs
   temporaryImage(ind)=1
   elements = sum(temp(:));
   if ((elements < 2^nbits) && (elements >= 2^nbits - M) then
      framenumber = framenumber + 1;
      [bitstream] = encodeEventsInImage(temporaryImage,currentXs,currentYs,nbits);
      outputimage(framenumber) = temporaryImage;
      writePNGImagetodisk(temporaryImage)
      writeLOCOImagetodisk(temporaryImage)
   end if
end for
```

Figure 17. *Algorithm for generating and encoding event location images*

Although this method does not result in higher compression ratios compared to the baseline strategy, it has some notable characteristics. This method enables us to transmit images to the decoder, which can help identify the type of dataset being dealt with. To maintain a standard for compression ratios, a constant value for nBits was used. It is worth noting that the bits per event required to compress x and y coordinates using this method are larger than the baseline compression due to the requirement to encode images instead of a stream of numbers. However, some compression ratios achieved using this method are comparable to those achieved using spike coding, such as the night-traffic, pendulum, and game datasets, which require more than one bit to encode the location image. Spike coding uses a similar method to encode datasets where bits required to encode x and y after merging are large. Since this thesis only discuss-

es lossless compression strategies, image codecs like JPEG were not used. Although JPEG is one of the most popular image codecs used to compress images, it is a lossy compression algorithm that reduces output image quality to achieve a higher compression ratio.

Sequence	Intermediate images bits per event using Loco	2^Nbits for encoding	Length of block (bits per event)	Total bits per event for encoding XY (loco)	Total bits per event for encoding timestamps	Total bits per event for encoding polarity	Total bits per event	Compression Ratio
Waterdrop	0.0247	11	0.0002	11.0249	0.1438	0.0533	11.222	5.7030
Lighter	0.6138	11	0.0009	11.6147	0.4034	0.1364	12.1545	5.2655
Football	0.2218	11	0.0005	11.2223	0.4157	0.2811	11.9191	5.3695
Jump	0.9311	11	0.0021	11.9332	1.1607	0.3602	13.4541	4.7569
Game	1.9200	11	0.0029	12.3372	1.6497	0.5172	14.5041	4.4125
Pendulum	1.0362	11	0.0135	12.0497	6.4774	0.7291	19.2562	3.3239
Pedestrians	1.4985	11	0.0094	12.5079	0.9660	0.1863	13.6602	4.6851
Daytime-traffic1	0.0638	11	0.0003	11.0641	1.5155	0.2862	12.8658	4.9744
Daytime-traffic2	0.0548	11	0.0006	11.0554	1.5495	0.2146	12.8195	4.9923
Night-Roadside	0.9726	11	0.0014	11.9740	1.8781	0.5059	14.358	4.4574
Night-Traffic	1.0233	11	0.0014	12.0247	2.1480	0.4056	14.5783	4.3901
Intersection	0.7358	11	0.0010	11.3193	0.0658	0.0406	11.4257	5.6014
Fluorescent	0.0882	11	0.0002	11.0884	0.1361	0.1493	11.3738	5.6269
Total Average								4.8673

Table 11. *Total bits for encoding all channels of data by encoding event images*

3.3 Using an occupancy matrix to encode x and y coordinates

In this method proposed in the thesis, x and y coordinates are encoded using an occupancy matrix and generating a bitstream after encoding the matrix. The method employs a function that encodes the occupancy of a grid of cells using arithmetic coding. The input for this method includes arrays that contain the row and column indices of the occupied cells in the grid. The output of this method includes arithmetic code lengths of the occupancy and the bitstream of the occupancy. The timestamps and polarity are compressed using Bzip2 and PPmD, respectively.

To understand the method, one needs to understand the concept of causal templates. A causal template is a technique for encoding time-series data in a compressed format. Time series refer to a sequence of data points measured over time, such as stock prices, weather data, or sensor readings. In many cases, the data points in a time series exhibit some degree of correlation with the previous data points, which means that the value of each data point can be predicted to some extent based on the previous data points. A causal template takes advantage of this correlation by encoding each data value as “the difference between the actual value of the data value and a predicted value” based on the previous data points [21]. This difference is called the "residual" or "prediction error".

In other words, instead of encoding each data point as a standalone value, a causal template uses a model to predict each data point based on the previous data points, and then encodes only the difference between the predicted value and the actual value. This approach can be more efficient than encoding each data point independently, especially for data sets with high correlation between adjacent data points. The use of causal templates can result in more efficient compression of time-series data by exploiting the correlation between the data points.

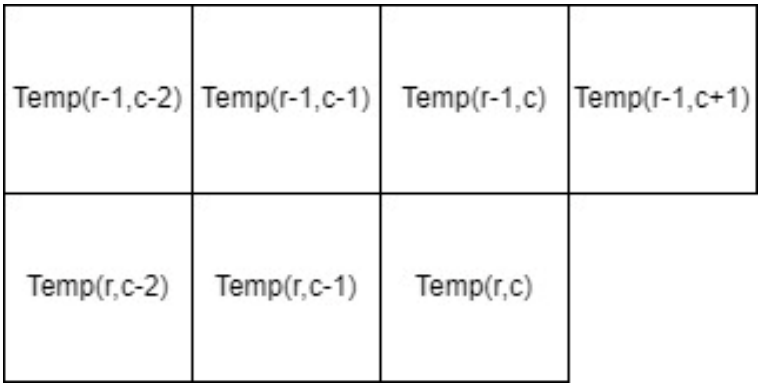


Figure 18. *Example of a causal template in a single iteration of a loop, ‘r’ represents rows and ‘c’ represents columns*

In the context of data compression, an occupancy matrix is a technique used to encode sparse data in a compressed format. Sparse data is characterized by having a majority of its values as zero or empty. To compress such data, it is often more efficient to encode only the non-zero entries rather than the entire matrix. An occupancy matrix is a data structure that represents the locations of

the non-zero entries in a sparse matrix. It is usually implemented as a binary matrix where each element indicates whether the corresponding element in the original matrix is non-zero (i.e., occupied) or zero (i.e., unoccupied). To compress a sparse matrix using an occupancy matrix, the binary matrix is first encoded using a suitable compression algorithm (such as Huffman coding or arithmetic coding). Then, the non-zero entries of the original matrix are encoded using a separate compression algorithm, such as run-length encoding, variable-length encoding, or delta encoding. During decoding, the occupancy matrix is first decoded to determine the locations of the non-zero entries in the original matrix. Then, the non-zero entries are decoded and placed back into the original matrix using the previously determined locations.

Incorporating the blocking mechanism discussed in section 3.2, this method uses the size of the block to determine its performance. Specifically, the block size determines the number of events taken into account to construct an occupancy matrix. To generate the matrix, the algorithm initializes a temporary matrix 'temp' of dimensions (numrows+2) x (numcols+3) with zeros. Then, for each occupied cell in the grid, the corresponding element in 'temp' is set to 1, forming a 2D occupancy matrix by adding 1 to 'temp'. This matrix consists of 1s for empty cells and 2s for occupied cells. Next, the occupancy of each cell is encoded using arithmetic coding. To accomplish this, a causal template is constructed for each cell with six neighbouring cells (three from the row above and three from the same row) as context for arithmetic coding. The probability of the current cell being occupied or empty is computed based on the context and frequency count of the context and occupancy. Using this probability, the arithmetic code length 'CL' of the occupancy is calculated, and the frequency count is updated accordingly.

In the final step of the algorithm, arithmetic coding is used to create a bitstream by encoding the occupancy status of each occupied cell. The linear index of the occupied cells is first determined and then the probability of each cell's occupancy is calculated using the frequency count of the corresponding cell. The arithmetic code length 'CLB' of the bitstream is calculated using the probability and the frequency count is updated accordingly. To evaluate the method, exper-

iments were conducted on the PKU dataset using 3 different block lengths, and the results are presented in Table 12.

In comparison to encoding location images for x and y coordinates, it is observed that this method delivers better performance on average. This is because occupancy matrices are more efficient at compressing sparse data, and the data being compressed has a highly predictable value. There is no discernible relationship between the block size and the number of bits required for compression, although for some datasets, such as jump and night-traffic, larger block sizes result in fewer bits required for compression. Although the improvement from using a larger block size is minor, it is still beneficial since larger block sizes generate fewer frames overall. As a result, there is less data to transmit, and since lossless compression methods are employed, the final size of the data remains unchanged. This method is comparable in terms of compression ratio to the baseline method and performs better for datasets with a single moving object generating fewer events in the background, such as lighter, jump, and game. However, for datasets such as daytime traffic and intersection, which include the movement of numerous objects generating events in several locations within a single frame, the baseline method produces better compression. It can be concluded that this method is suitable for scenarios where there is a stationary object and the camera remains fixed, but it is less effective when multiple objects are moving, and events are generated in numerous locations within one frame.

Sequence	Total bits per event for encoding XY Block length 10000	Total bits per event for encoding XY Block length 15000	Total bits per event for encoding XY Block length 20000	Total bits per event for encoding timestamps	Total bits per event for encoding polarity	Total bits per event	Compression Ratio
Waterdrop	0.08809	0.07898	0.08239	0.1438	0.0533	0.2794	228.9852
Lighter	2.1069	2.0943	2.0983	0.4034	0.1364	2.6341	24.2967
Football	3.4847	3.3323	3.3296	0.4157	0.2811	4.0264	15.8950
Jump	6.1656	6.125	5.9677	1.1607	0.3602	7.4886	8.5463
Game	5.7814	4.9563	5.7008	1.6497	0.5172	7.1232	8.9847
Pendulum	5.9196	5.8454	5.8135	6.4774	0.7291	1.,0200	4.9155
Pedestrians	5.4613	5.4043	5.5292	0.9660	0.1863	6.5566	9.7611
Daytime-traffic1	8.7903	8.8898	8.9206	1.5155	0.2862	10.5920	6.0422
Daytime-traffic2	3.7393	3.7804	3.7676	1.5495	0.2146	5.5034	11.6291
Night-Roadside	10.076	10.108	10.083	1.8781	0.5059	12.4600	5.1364
Night-Traffic	11.807	11.708	11.6600	2.1480	0.4056	14.2136	4.5027
Intersection	2.0533	2.0476	2.0449	0.0658	0.0406	2.1513	29.7494
Fluorescent	3.3607	3.2293	3.2106	0.1361	0.1493	3.4960	18.3066
Total Average							6.8880

Table 12. *Total bits for encoding all channels of data by encoding occupancy matrices*

3.4 Generating and compressing the optical flows of event data

Optical flow is the apparent motion of brightness patterns in an image. It is explained in more detail in section 2.4. Optical flow has numerous applications some of which are explained below:

1. Motion estimation: Optical flow can be used to estimate the motion of objects in a video sequence. This can be useful for tracking moving objects, detecting changes in the scene, and analysing the motion of crowds or vehicles.
2. Video compression: Optical flow can be used to compress video sequences by identifying and removing redundant motion information. Compression revolves around finding patterns and using a predictive algorithm to compensate for that recurring pattern. Optical flow helps identify such redundant patterns and makes compression more effective.

3. Autonomous navigation: Optical flow can be used to help robots navigate through an environment by providing information about the motion of objects in their field of view.
4. Augmented reality: Optical flow can be used to track the motion of the user's head or body in real-time, which can be used to create immersive augmented reality experiences.
5. Driver assistance systems: Optical flow can be used in driver assistance systems to detect the motion of other vehicles on the road, and to alert the driver if there is a potential collision. This is one of the most recent and interesting use cases of optical flow. Utilizing this method, we will be able to configure autonomous cars better.

The optical flow in event cameras was studied by selecting the DAVIS dataset, as the PKU dataset was limited to motion with a fixed background and a single or multiple moving objects. The DAVIS dataset was chosen as it contained sets of data where the camera was moving while the background remained fixed, as in the Slider dataset shown in Figure 19. This type of movement more closely resembles human motion, where the objects in the frames remain in the same place and only the sensor moves. Additionally, some datasets in the DAVIS dataset contain a moving background while the camera sensor is fixed.

Optical flow was calculated using a machine learning model trained on different sets of data from the same dataset. For this study, the EV-FlowNet model was chosen as it had a smaller average endpoint error (AEE) in experiments from Haagenars et al. [26]. The architecture of the model was shown in Figure 20. The input event was reduced in resolution using four encoder layers that used strided convolution. Then, the downsampled event was fed through two layers of residual blocks. After that, the activations were passed through four decoder layers that used upsample convolution, and they were connected with the corresponding encoder layer using skip connections. Each decoder activation set was also “passed through a depthwise convolution layer to generate a prediction” of the flow. A loss was applied to this flow prediction, and it was concatenated with the decoder activations [28].

In this study, the generated flow was extracted by the model and used for motion compensation. The following datasets were used - boxes 6DOF, calibration, dynamic 6DOF, office zigzag, poster 6DOF, shapes 6DOF, and slider depth. Out of these, poster 6DOF and boxes 6DOF were too large to fit in memory and were thus removed from the final results. Flows and predictions from those sets were still calculated.

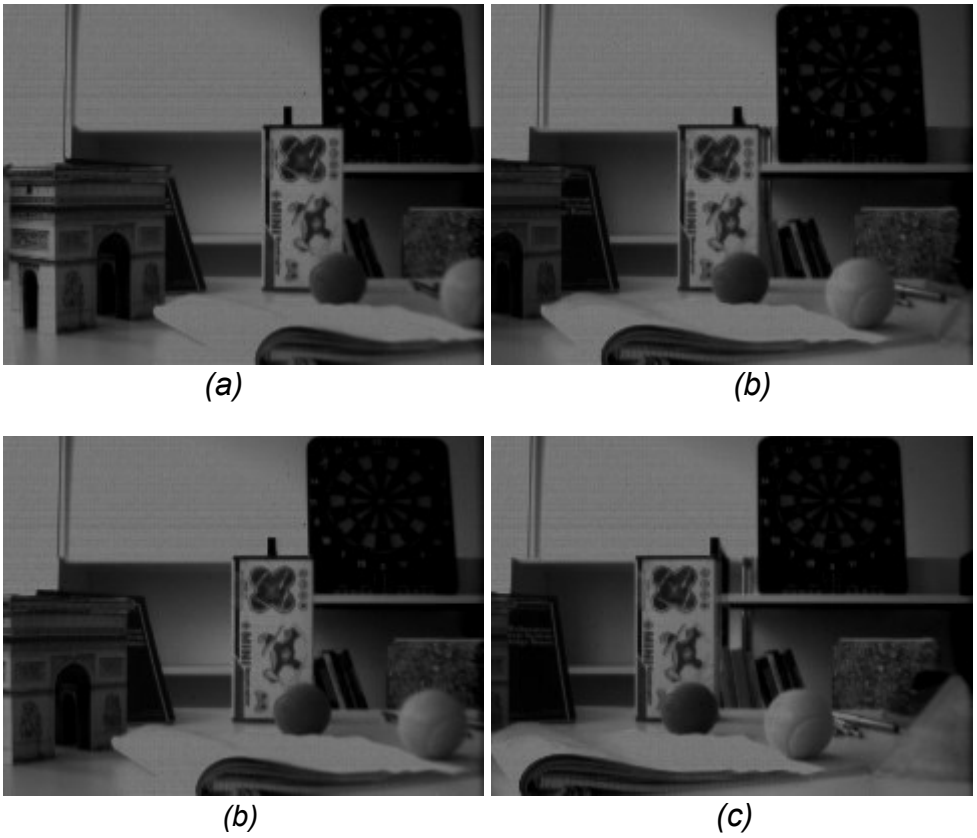


Figure 19. *Frames 1 (a), 5 (b), 10 (c) and 15 (d) from Slider Depth, DAVIS dataset*

To understand the meaning of colors in a flow image, it is necessary to understand the representation of colors first. A visual aid that helps to comprehend the color scheme used in flow images is shown in Figure 21. The flow images and their interpretation are presented later in the thesis.

After running the model on all the datasets, the resultant optical flows are shown below. A lot of frames were generated with characteristic flows and some noteworthy and unique flows are show below. Original image frames in black and white are shown side by side to understand the direction of flow.

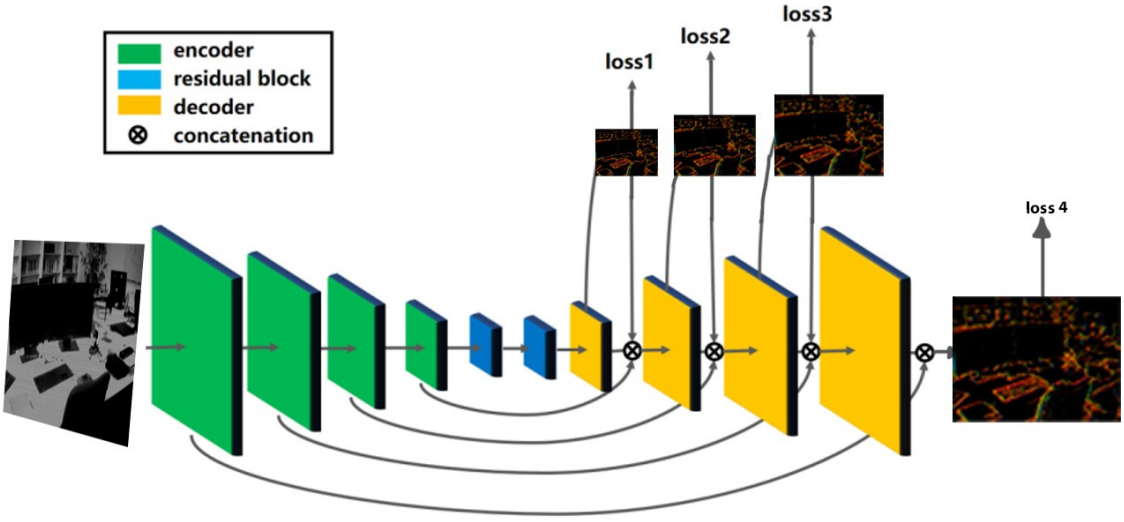


Figure 20. *EV-FlowNet architecture [28]*

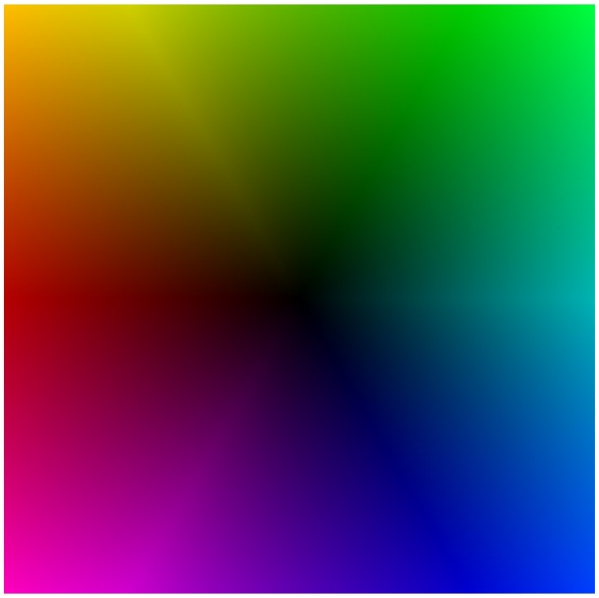


Figure 21. *Flow-map Colours. These colours depict the direction of the optical flow [4]*

Based on figure 21, it can be understood that the movement of pixels in figure 22 from frame 44 to 45 was in the direction of left to right. Similarly, most pixels moved upwards to the right when shifting from frame 271 to 272.

In optical flow, there are two components, horizontal and vertical, which combine to form the complete flow, as depicted in Figure 22. In this thesis, the author chose to separate the horizontal and vertical components and utilize them to deblur the image formed by aggregating events in spacetime. Each flow

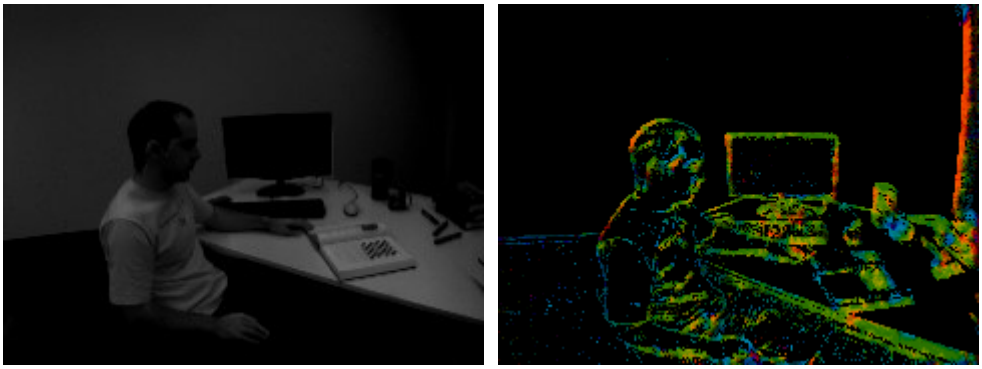
component is an array of values, with 240 arrays for horizontal and 180 arrays for vertical in the case of the DAVIS dataset's 240x180 pixel resolution camera. Each value in the array represents the movement of a pixel in the direction of the respective flow. For instance, a positive value of two for the horizontal component means that a pixel at that location moved right by two-pixel widths when moving from one frame to the next. Figure 23 provides an example of these components. Since flow values are typically quite small, a scaling factor of 128 is applied to each component to enhance the effect of the flows during deblurring. This value is consistent with the work of Hagenaars et al. [28].

The normalization of timestamps to a local timeline is crucial as the neural network processes frames sequentially. However, using event cameras for optical flow estimation poses a challenge because events are not evenly distributed in time, and nearby events may have significantly different timestamps, resulting in inaccurate motion estimation. To address this issue, we used timestamp normalization, which transforms the event stream such that events within a specific time window are considered to have occurred simultaneously. We adopted the same normalization formula as Hagenaars et al. in [28]:

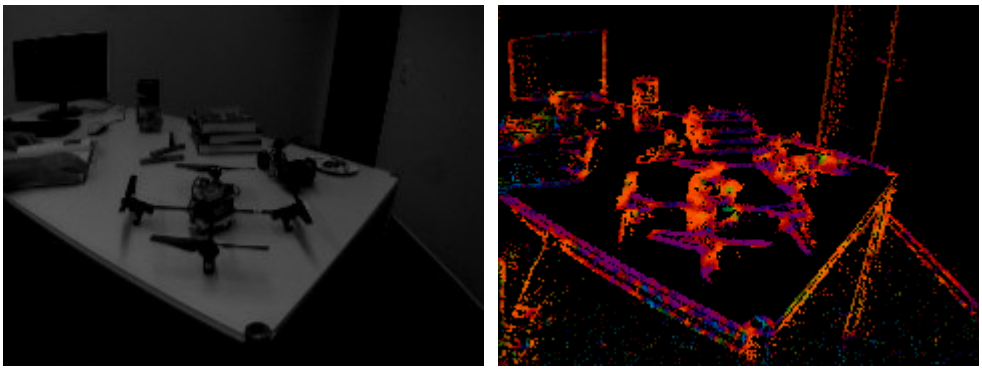
$$normalized\_timestamp = (ts - ts[0]) / (ts[-1] - ts[0]) \quad (3)$$

where, ts denotes the current timestamp, ts[0] is the first timestamp value for the dataset and ts[-1] denotes the last timestamp value in the dataset. Using this method, a set of timestamps that fall in the range of zero to one was obtained.

(a) (b)



(c) (d)



(e) (f)

Figure 22. *Dynamic-6DOF original frames 44(a), 271(c) and 577(e). Generated optical flows using EV-FlowNet model for the same frames (b), (d) and (f) respectively.*

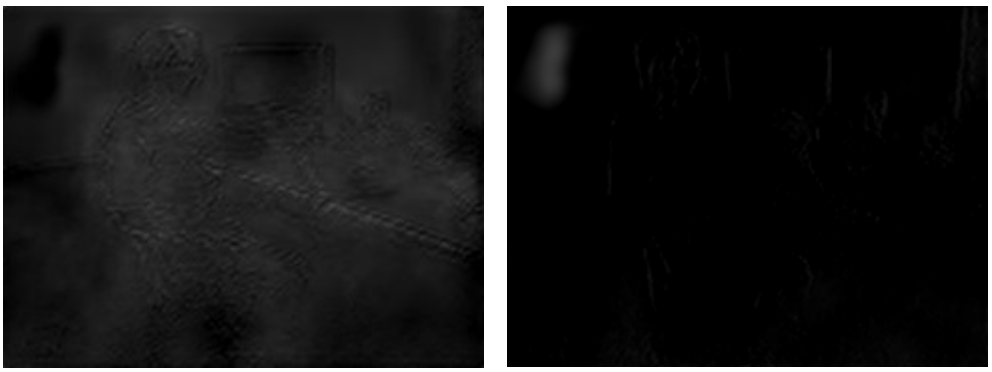


Figure 23. *Horizontal (left) and vertical (right) components of optical flow for frame 44 of Dynamic 6DOF. Each flow component is upscaled by a factor of 2560 to be visible (only for this image)*

In order to evaluate the effectiveness of the deblurring process, it is important to have an understanding of what the original images look like. To create an original frame, a temporary image is generated with the same dimensions as the DAVIS dataset images, which is 240x180 pixels. Initially, this image is filled with zeroes to create a completely black image. The locations of event occurrences are then identified from the event stream and these coordinate values are changed to one. This process is repeated until a predetermined number of events are plotted, which is 15000 in this case. The resulting black and white image closely resembles the original image. The total number of distinct locations that were plotted is also calculated, which is a useful metric for evaluating the effectiveness of the deblurring process. Several examples of such images are shown in Figure 24 and 25. Two different datasets are show in these figures. Figure 24 shows frame from Slider-depth which only exhibits linear motion. While this is a characteristic motion to observe, figure 25 shows frames which have non-unfirm motion. Both different types of motions are observed for and presented in this work for better understanding and application of this method. Other datasets comprise of more different kinds of movements.

“Using the contrast maximization proxy loss technique for motion compensation” as described in [26], the optical flows are estimated. This approach aims to learn to accurately estimate optical flow by leveraging the “spatiotemporal misalignments between the events triggered by the same portion of a moving edge”, which can be interpreted as a form of blur [28]. To retrieve accurate optical flow

information, it is necessary to compensate for this motion, effectively deblurring the event partition.

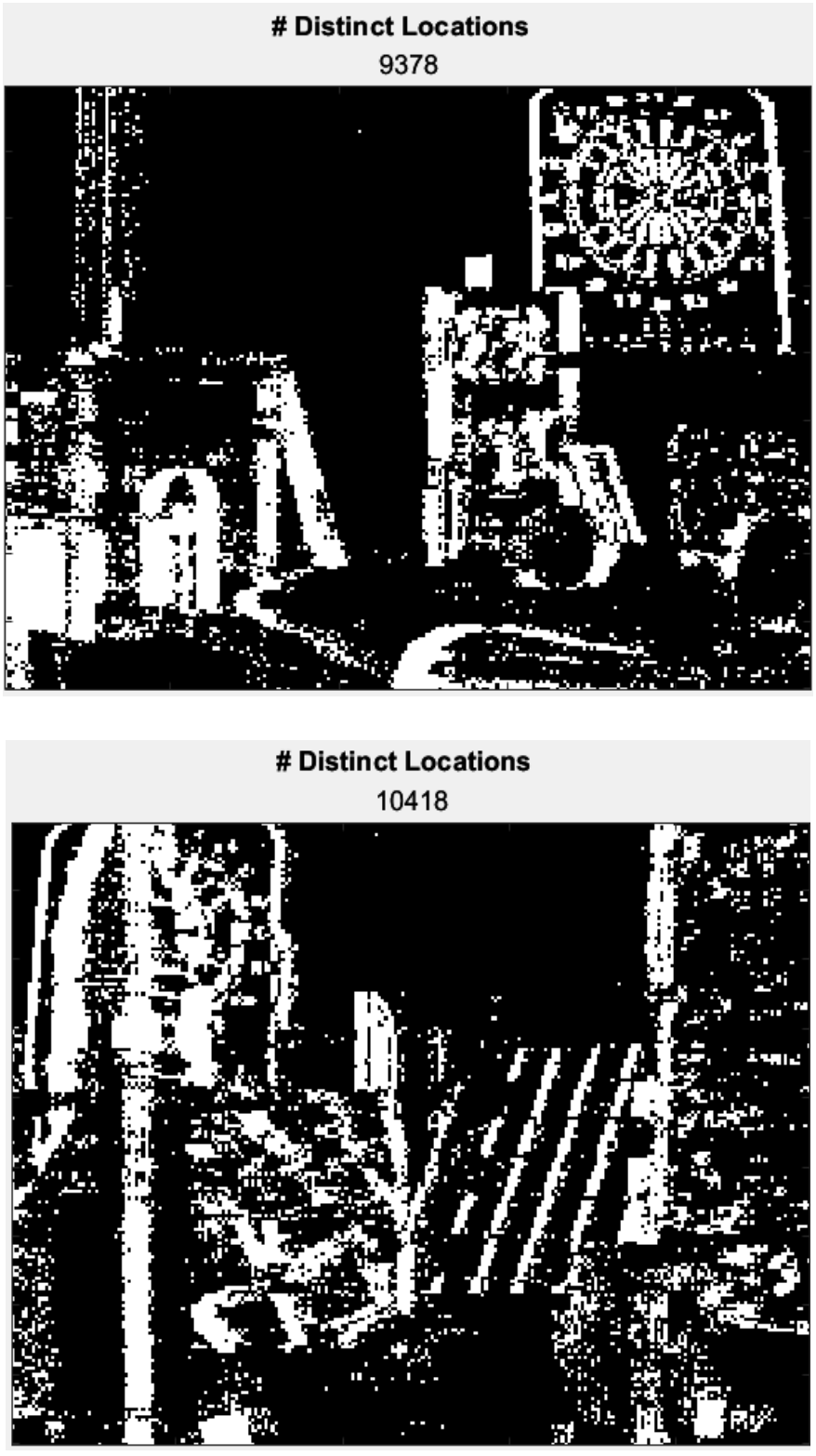


Figure 24. *Frames 1(top) and 69(bottom) from Slider Depth formed by accumulating event location on a dark image.*

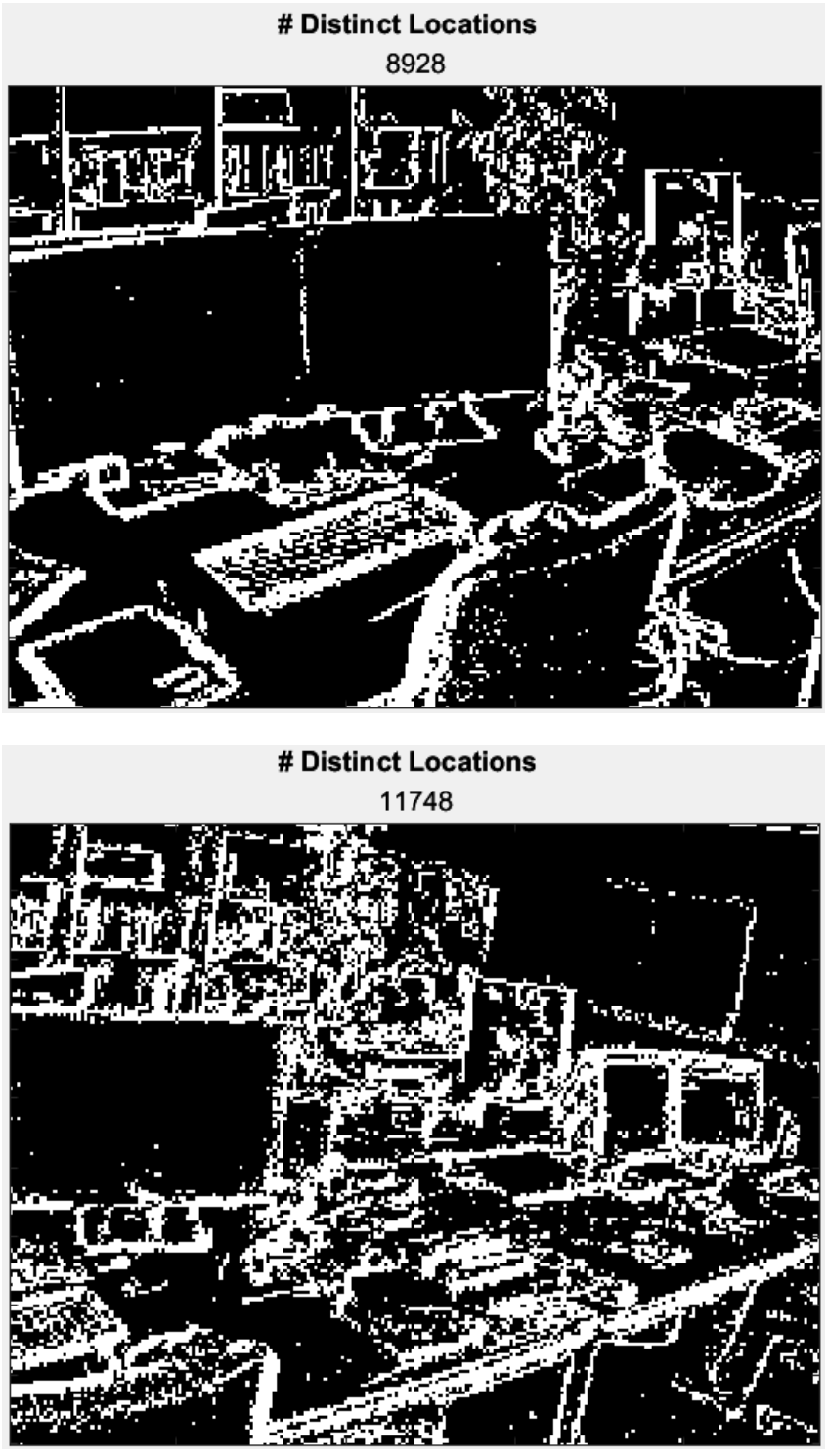


Figure 25. *Frames 1(top) and 71(bottom) from Office Zigzag formed by accumulating event location on a dark image.*

The per-pixel optical flow $\boldsymbol{u}(x) = (u(x), v(x))^T$ is used to propagate the events to a reference time t_{ref} using formula 2.

To compensate for blur, we utilized formula 4 to generate new images by aggregating event locations. The resulting images, along with the number of distinct locations included in them, can be seen in Figure 26 and 27.

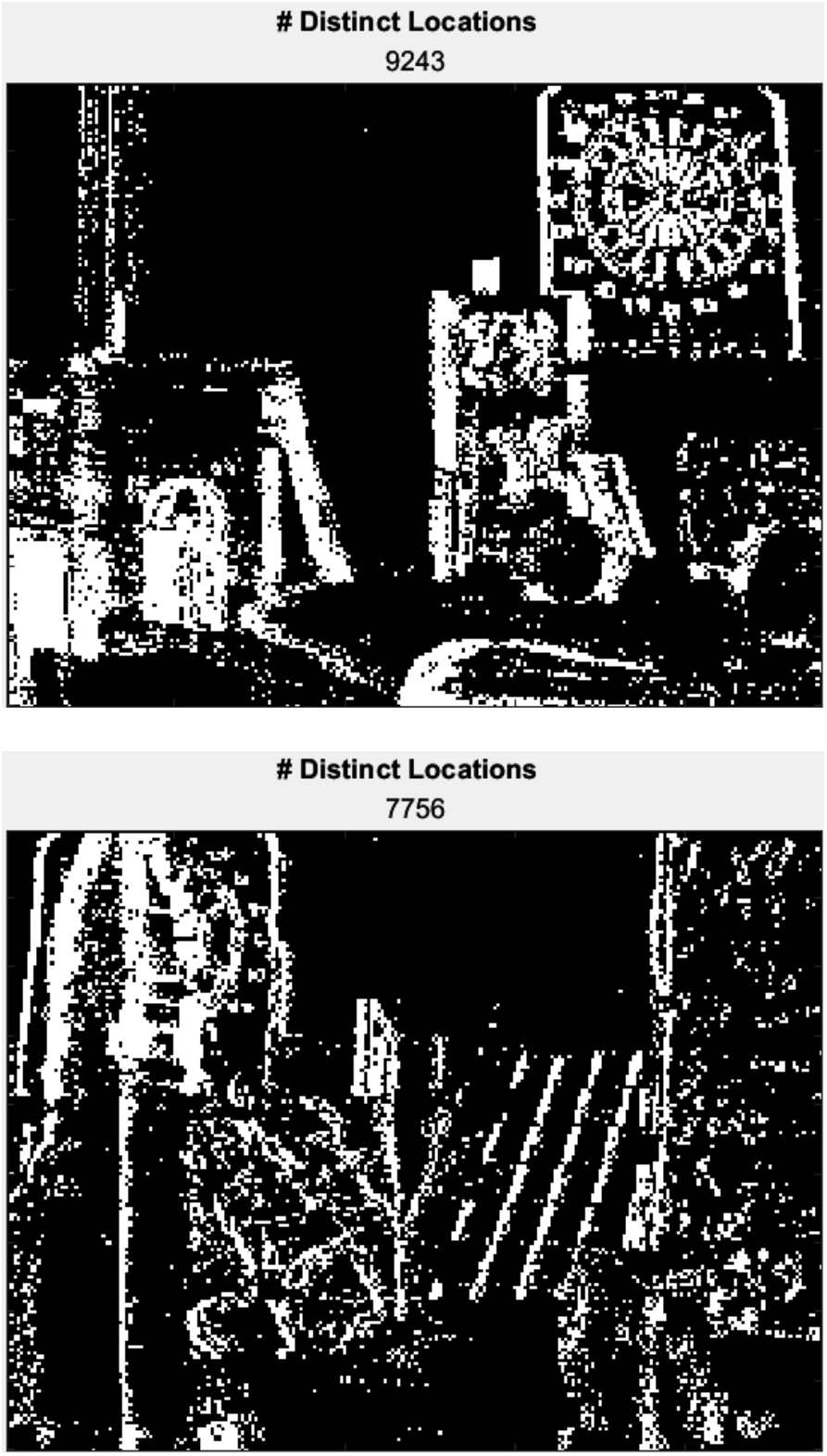


Figure 26. *Deblurred images for frames 1(top) and 69(bottom) from Slider Depth after compensating for blur using scaled flows*

In comparing the images in figure 26 to those in figure 24, we observe that the number of distinct locations is reduced significantly, with 135 for frame 1 and 2662 for frame 69. When an image is blurred, neighbouring pixels become more similar to each other, which results in a loss of information and detail.

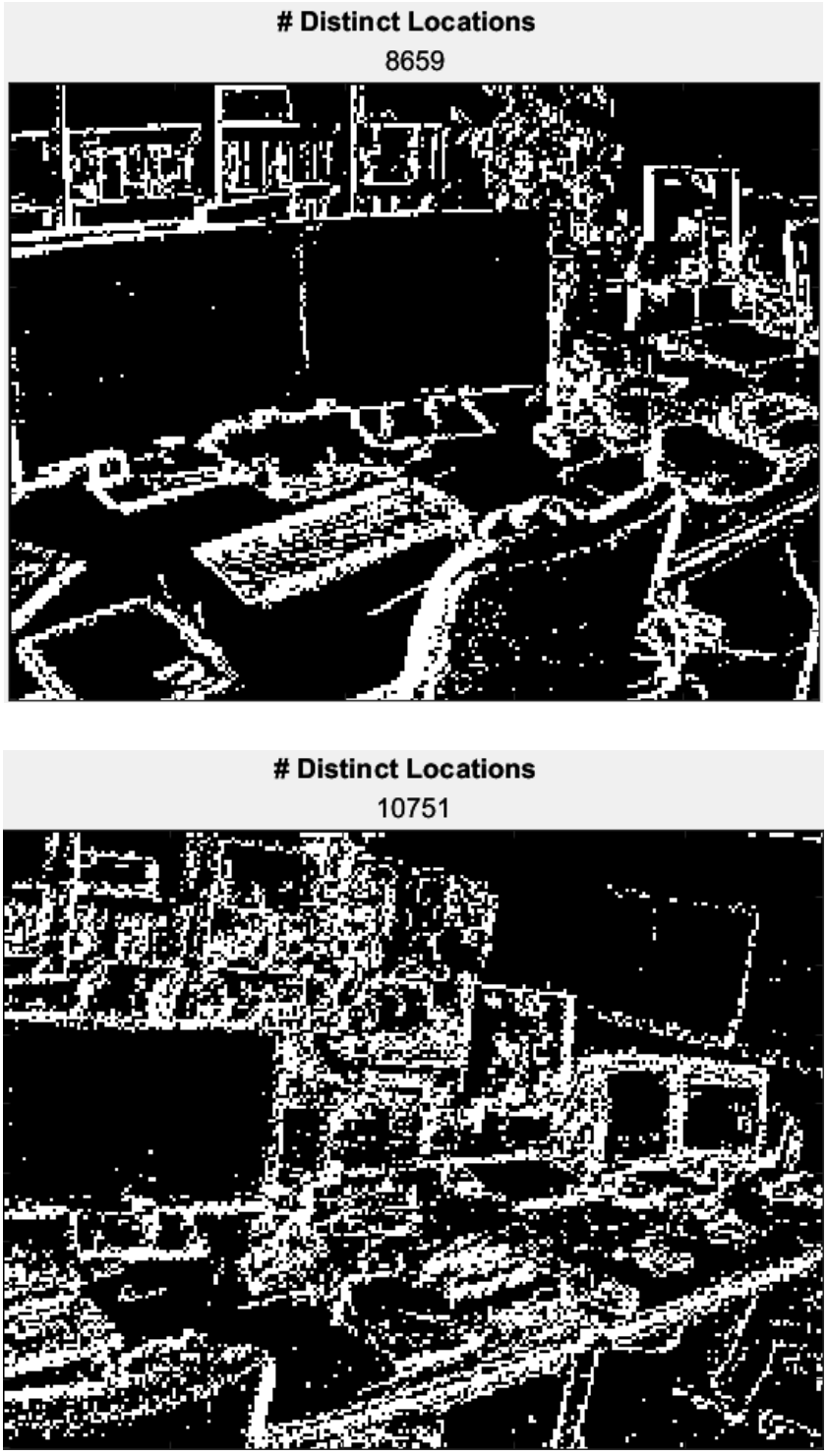


Figure 27. *Deblurred images for frames 1(top) and 71(bottom) from Office Zig-zag after compensating for blur using scaled flows.*

Therefore, deblurring the image sharpens the edges and details, making them more distinct, and this can reduce the number of distinct locations. In simpler terms, when an image is blurred, the edges and details in the image become less sharp, resulting in more distinct locations.

Secondly, the edge width of objects in an image reduces after deblurring. This is because a blurred image is not as sharp as a deblurred image, causing edges of objects to appear thicker due to the apparent motion. Once the motion has been compensated for, fewer pixels are needed to mark an edge. Blurring tends to spread out the edges in an image, making them wider and less distinct. In contrast, when the blur is removed through deblurring, the edges become sharper and more distinct, resulting in a reduction in edge width. Blurring causes neighbouring pixels to become more similar, resulting in a smoother gradient across the edge, which leads to a wider transition region between the edge and the background. Deblurring the image removes this smoothing effect, making the gradient steeper and the edge more distinct, resulting in a narrower transition region and a reduction in edge width.

Moreover, regions with too many events are more distinct after deblurring, which adds more detail to the image and allows for better object recognition. Since there are fewer distinct locations after deblurring, compressing and encoding this data is much more efficient and results in a smaller output size. Smaller compressed file sizes are much faster to transmit and ease any other process that need to be performed over them.

In order to achieve more efficient compression ratios by decreasing the number of distinct locations, an additional step of quantization was performed on the scaled optical flows. Quantization is a commonly used technique for reducing the amount of data required to represent a signal or image by decreasing the number of bits allocated to represent each value. This can lead to a loss of quality in the resulting signal. Given the small magnitude of the flow values, the loss of detail was negligible. Quantization was performed by determining the minimum and maximum values for each component of the flow and applying a formula to each value of the flow.

$$quantized\_flow = round(w - wmin)/step\_size \quad (5)$$

where, step size is calculated as the difference between the largest and smallest value of the flows divided by 255.

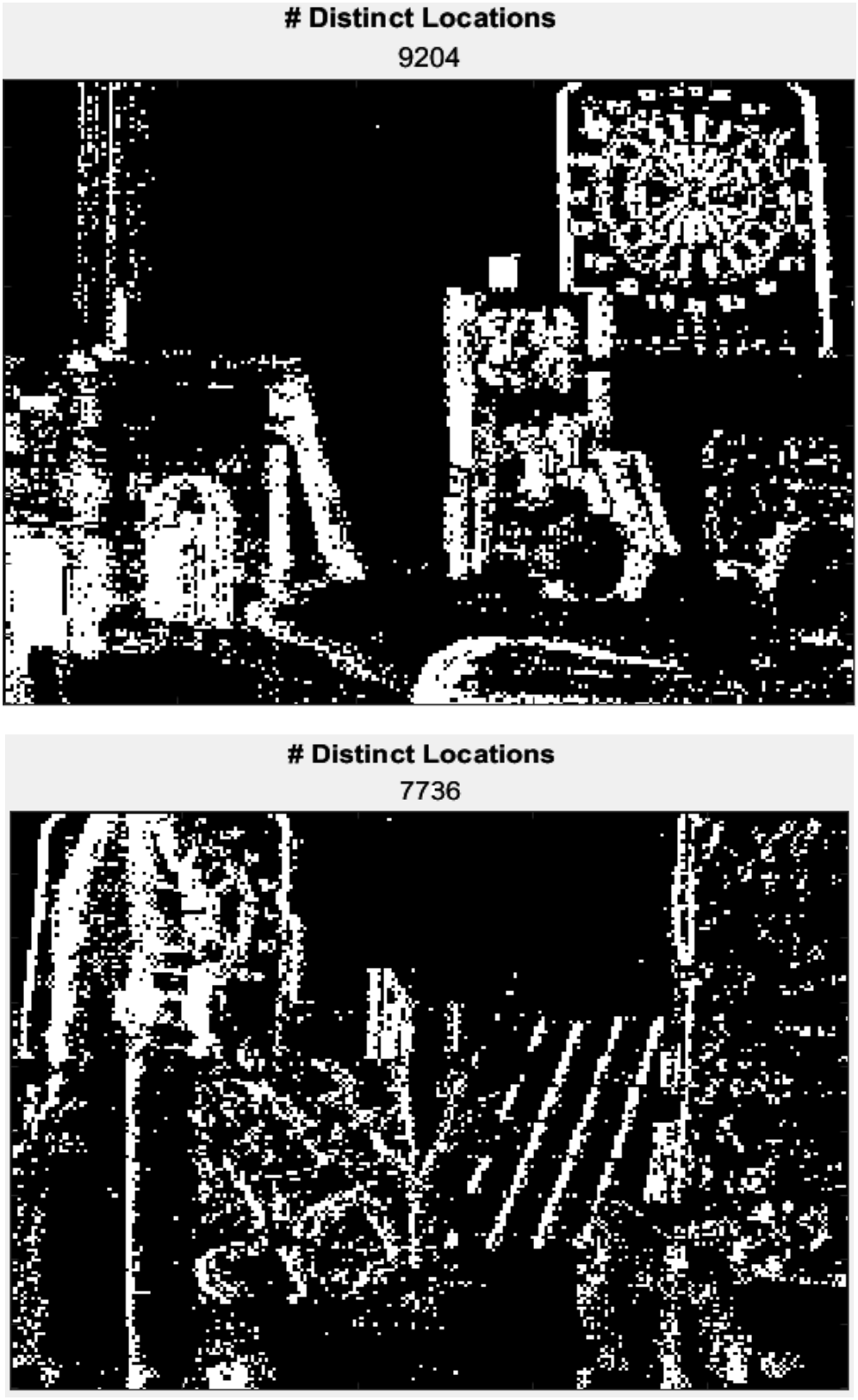


Figure 28. *Deblurred images for frames 1(top) and 69(bottom) from Slider Depth after compensating for blur using quantized and scaled flows.*

Quantization is a technique that can be used to reduce noise effects in the flow. By rounding the pixel values to the nearest quantization level, the effects of small variations or errors in the flow component can be reduced. Optical flow-based deblurring is a computationally intensive process, particularly for large images. However, by quantizing the flow, the number of bits required to represent each pixel value can be reduced, resulting in a smaller data size and reduced computational complexity. Additionally, quantization can improve the convergence of the optical flow algorithm by reducing the range of possible pixel values, which may result in faster and more accurate convergence. Furthermore, quantization can make the deblurring process more robust to small variations in the image or noise in the optical flow estimates, resulting in better quality restorations, particularly in cases where the input image is noisy or contains significant motion blur.

After quantization, the quantized flow components can be used for deblurring, similar to the process used for scaled flows. The results of this experiment are presented in figures 28 and 29. Quantized flows can further decrease the number of distinct locations, aiding in decreasing the time and complexity required for encoding locations during flow transmission. Although quantization does not provide a substantial decrease in distinct locations compared to scaled flows, the minor reduction could save a few bits during encoding. The comparison of figures 26 and 28 reveals that no information is lost, and the objects represented in scaled deblurring are identical to those in quantized deblurring.

Based on the results of the experiment, it can be concluded that quantized flows provide the best deblurring outputs and the greatest reduction in the number of distinct locations without sacrificing information. Therefore, in this thesis, we encode and compress the quantized flows for transmission. As the flows can be visualized as images after scaling, an image codec is an appropriate choice for encoding the flows. In this study, JPEG-LS, PNG, and JPEG 2000 (JP2) were selected. Table 15 presents the original size of the generated optical flows, and Table 16 summarizes the compression results. The average size of optical flows is calculated as the sum of the sizes of the horizontal and vertical components, which is then multiplied by the number of frames generated to produce the total average size of flows. Notably, all datasets in the experiment have

similar sizes for a single flow, as the number of values in a flow component is dependent on the DAVIS sensor's resolution.

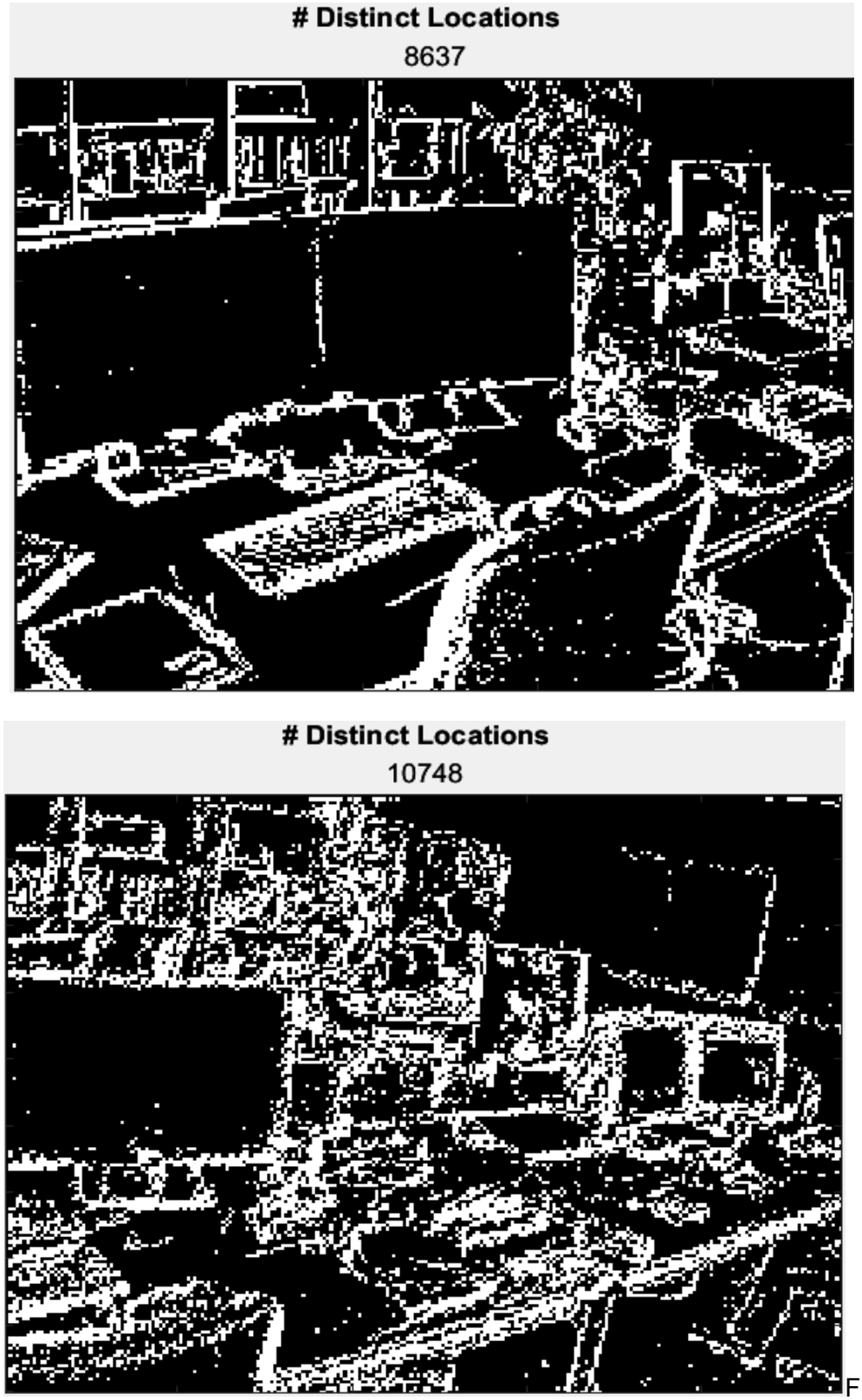


Figure 29. *Deblurred images for frames 1(top) and 71(bottom) from Office Zig-zag after compensating for blur using quantized and scaled flows.*

Sequence	Number of optical flow images generated	Average size of optical flow for 1 frame (bytes)	Total size of all flows (bytes)
Slider_Depth	70	2 193 433	153 540 310
Shapes_6DOF	1196	2 194 831	2 625 017 876
Dynamic_6DOF	3810	2 196 984	8 370 509 040
Calibration	1421	2 191 965	3 114 782 265
Office_zigzag	514	2 193 754	1 127 589 556

Table 13. *Original optical flow sizes for comparison, DAVIS dataset*

Sequence	Average size of flow compressed using JP2 (bytes)	Total size of flows compressed using JP2 (bytes)	Average size of flow compressed using JPEG-LS (bytes)	Total size of flows compressed using JPEG-LS (bytes)	Average size of flow compressed using PNG (bytes)	Total size of flows compressed using PNG (bytes)	Compression Ratio
Slider_Depth	11,127	778,890	**10,111**	**707,770**	13,080	915,600	216.9353
Shapes_6DOF	12,319	14,733,524	**11,847**	**14,169,012**	14,048	16,801,408	185.2647
Dynamic_6DOF	12,795	48,748,950	**12,014**	**45,773,340**	14,959	56,993,790	182.8686
Calibration	11,429	16,240,609	**10,619**	**15,089,599**	13,962	19,840,002	206.4191
Office_zigzag	11,547	5,935,158	**10,477**	**5,385,178**	13,071	6,718,494	209.3876

Table 14. *Comparison of compressing quantized optical flows using image codes, best in bold*

The compression ratio in table 14 is calculates by dividing the total size of all flows (table 13) by the total size of compressed optical flow. It can be concluded that JPEG-LS performs the best when compared to the other image codes. Thus, the compression ratio is calculated using the size of the compressed flows by JPEG-LS.

In the context of image compression, JPEG-LS is a versatile format that can achieve both lossless and lossy compression. It can preserve image quality and data integrity, or compress images with some degree of quality loss, depending on the desired compression ratio. PNG, on the other hand, is strictly a lossless format that can only compress images without any loss of data. JP2 is also a strong contender, but its algorithm is more complex than JPEG-LS, making it less desirable in situations where a simpler solution is preferred. JPEG-LS is particularly suitable when there is a need to balance file size with image quality. Additionally, JPEG-LS supports up to 16 bits per colour channel, whereas PNG

and JP2 support up to 48 and 38 bits per pixel respectively. Hence, JPEG-LS is the better choice when dealing with high-quality images that have a lower colour depth.

4. CONCLUSION

In this thesis, we have explored the problem of lossless compression of event data and optical flow images from event cameras. Event cameras are a relatively new type of vision sensor that can capture fast and dynamic visual scenes with high temporal resolution and low power consumption. However, the high data rates and sparse nature of event data make it challenging to store and transmit the data efficiently. Therefore, the goal of this work was to develop efficient compression algorithms that can compress the event data and optical flow images while preserving their important features.

In the first part of the thesis, we proposed a lossless compression algorithm for event data that is based on isolating event streams and compressing them individually using existing compression algorithms. The proposed algorithm was at par with state-of-the-art methods in terms of compression ratio and reconstruction accuracy on a publicly available dataset. We also demonstrated that our algorithm could achieve a high compression ratio while preserving the spatio-temporal properties of the events, which are important for applications such as object detection and tracking.

In the second proposed method, we tried to create location image using x and y coordinates from the event data and compressed these using image codecs. Even though the compression ratio in this case was not as high as the first method, we identified a novel to compress event data and in which particular case it could be used.

In the third method, we discussed and compressed event data using occupancy matrices and causal templates. We discovered an interesting approach to data compression and a temporally efficient method to do so.

In the final part of the thesis, we focused on the compression of optical flow images generated from event data. We used an encoder-decoder style machine learning model to generate optical flows. These flows were then used to deblur images that were generated by accumulating event data on an image. Flows were then further quantized to reduce the number of occupied pixels in an im-

age saving some cost of compression. The proposed method achieved a high compression ratio while preserving the accuracy of the optical flow fields, which are essential for tasks such as scene reconstruction and navigation.

Overall, our results demonstrate the feasibility and effectiveness of lossless compression of event data and optical flow images from event cameras. This study confirmed that using methods like PPmD and LZMA are ideal choices when compressing raw event data. This work opens new possibilities for the deployment of event cameras in real-world applications. Using optical flows for these use cases like navigation and autonomous driving will help take the industry forward.

REFERENCES

[1] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, D. Scaramuzza, Event based Vision: A Survey, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 1, 2022, pp. 154-180.

[2] N. Khan, K. Iqbal, M. G. Martin, Lossless Compression of Data From Static and Mobile Dynamic Vision Sensors-Performance and Trade-Offs, IEEE Access, vol. 8, 2020, pp. 103149-103163.

[4] A. Z. Zhu, L. Yuan, K. Chaney, K. Daniilidis, “Unsupervised Event-based Learning of Optical Flow, Depth, and Egomotion”, 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Long Beach, CA, USA, 2019, pp. 1694-169.

[5] A. Tony, A. Paigwar, Khushdeep S. Mann, A. Renzaglia, C. Laugier, Fusing Event-based and RGB camera for Robust Object Detection in Adverse Conditions, 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 2022, pp. 933-939.

[6] K. Iqbal, N. Khan and M. G. Martini, "Performance Comparison of Lossless Compression Strategies for Dynamic Vision Sensor Data," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 2020, pp. 4427-4431.

[7] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. D. Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, A low power, fully event-based gesture recognition system, IEEE Conf. Comput. Vis. Pattern Recog. (CVPR), 2017.

[8] Z. Bi, S. Dong, Y. Tian and T. Huang, Spike Coding for Dynamic Vision Sensors, 2018 Data Compression Conference, Snowbird, UT, USA, 2018, pp. 117-126.

[9] S. Tsai Sam, D. Chen, G. Takacs, V. Chandrasekhar, M. Makar, R. Grzeszczuk, B. Girod, Improved Coding for Image Feature Location Infor-

mation, Proceedings of SPIE - The International Society for Optical Engineering. 10.1117/12.935619.,2019

[10] S. P. Strong, R. Koberle, Rob R. de Ruyter van Steveninck, W. Bialek, Entropy and information in neural spike trains, Physical review letters, 1998, pp. 80-197.

[11] T. Nishimoto and Y. Tabei, LZRR: LZ77 parsing with right reference, Proc. Data Compress. Conf. (DCC), Mar. 2019, pp. 211–220.

[12] X. Zhao and B. Li, Implementation of the LZMA compression algorithm on FPGA, 2017 International Conference on Electron Devices and Solid-State Circuits (EDSSC), Hsinchu, Taiwan, 2017, pp. 1-2.

[13] Y. Collet, Murray S. Kucherawy. Zstandard Compression and the application/zstd Media Type.” RFC 8878, 2018, pp: 1-45.

[14] J. Gailly, M. Adler, Zlib compression library, 2004

[15] J. Alakuijala and Z. Szabadka, Brotli Compressed Data Format, Internet Engineering Task Force, document RFC 7932, 2016.

[16] Y. Collet. Finite State Entropy, 2019 [Online]. Available: https://github.com/Cyan4973/FiniteStateEntropy

[17] D. Blalock, S. Madden, and J. Guttag, Sprintz: Time series compression for the Internet of Things, Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol., 2018, vol. 2, no. 3, p. 93.

[18] D. Lemire and L. Boytsov, Decoding billions of integers per second through vectorization, Software Practice and Experience, vol. 45, no. 1, 2015, pp. 1–29..

[19] S. H. Gunderson, Snappy: A Fast Compressor/Decompressor, 2015, Available: *https://github.com/google/snappy*

[20] National Engineering Laboratory for Video Technology (NELVT) and Institute of Digital Media (Peking University). PKU-DVS Dataset. Accessed: Oct. 1, 2019. [Online]. Available: *https://pkuml.org/resources/pku-dvs.html*

[21] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, D. Scaramuzza, The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM, Int. J. Robot. Res., vol. 36, no. 2, 2017, pp. 142–149.

[22] N. Khan and M. G. Martini, ‘‘Bandwidth modeling of silicon retinas for next generation visual sensor networks,’’ Sensors, vol. 19, no. 8, 2019, p. 1751.

[23] N. Khan, K. Iqbal, and Maria G. Martini, Time-Aggregation-Based Lossless Video Encoding for Neuromorphic Vision Sensor Data, IEEE Internet of things journal, vol. 8, January 1, 2021

[24] T. Brox, A. Bruhn, N. Papenberg, J. Weickert, High accuracy optical flow estimation based on a theory for warping. In European Conference on Computer Vision, Springer, Berlin, Heidelberg, 2004, pp: 25-36

[25] R. Benosman, S. Leng, C. Clercq, C. Bartolozzi, M. Srinivasan, Asynchronous frameless event-based optical flow, Neural Networks, 27, 2012, pp: 32–37

[26] Jesse J. Hagenaars, F. Paredes-Vallés, Guido C. H. E. de Croon, Self-Supervised Learning of Event-Based Optical Flow with Spiking Neural Networks, Advances in Neural Information Processing Systems, Volume34 – NeurIPS, 2021.

[27] G. Gallego, M. Gehrig, and D. Scaramuzza. Focus is all you need: Loss functions for event-based vis. In IEEE Conf. Computer Vision Pattern Recognition, 2019, pp: 12280–12289.

[28] Alex Z. Zhu, L. Yuan, EV-FlowNet: Self-supervised optical flow estimation for event-based cameras, Robotics: Science and Systems XIV. Robotics: Science and Systems Foundation, 2018.

[29] N. Ballas, L. Yao, C. Pal, A. Courville. Delving deeper into convolutional networks for learning video representations. Inernational Conference for Learning video representations, 2015.

[30] Alex Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, K. Daniilidis, The multivehicle stereo event camera dataset: An event camera dataset for 3D perception. IEEE Robot. And Autom. Lett., 3(3), 2018, pp:2032–2039.

[31] C. D. Gracia and S. Sudha, MePPM- Memory efficient prediction by partial match model for web prefetching, 2013 3rd IEEE International Advance Computing Conference (IACC), Ghaziabad, India, 2013, pp. 736-740.

[32] A. Anderson, J. Garland, Y. Wen, B. Barabasz, K. Persand, A. Vasudevan D. Gregg. Hardware and Software performance in deep learning. Many-more Computing: Hardware and Software, 2019

[33] I. Schiopu, R. C. Bilcu, Lossless Compression of Event Camera Frames, IEEE Signal Processing Letters, vol. 29, 2022, pp. 1779-1783.

[34] M. Almatrafi and K. Hirakawa, DAViS Camera Optical Flow, IEEE Transactions on Computational Imaging, vol. 6, 2020, pp. 396-407

[35] M. Liu and T. Delbruck, Adaptive Time-Slice Block-Matching Optical Flow Algorithm for Dynamic Vision Sensors, Abmof For Dynamic Vision Sensors, 2018

[36] D. Konstantinos, Z. Alex Zihao, Methods, Systems, And Computer Readable Media For Estimation Of Optical Flow, Depth, And Egomotion Using Neural Network Trained Using Event-Based Learning, The Trustees of the University of Pennsylvania, 2020

[37] A. K. Kosta, K. Roy “Adaptive-SpikeNet: Event-based Optical Flow Estimation using Spiking Neural Networks with Learnable Neuronal Dynamics”, Purdue University