

Dalal Manasi Sameer

LOSSLESS COMPRESSION OF DATA FROM DAVIS EVENT CAMERA

M. Sc. (Tech) Thesis
Information Technology and Communication
Prof. Ioan Tabus
Prof. Karen Eguiazarian
May 2024

ABSTRACT

Dalal Manasi Sameer: Lossless Compression of Data from DAVIS Event Camera
M. Sc. (Tech) Thesis
Tampere University
Master's Programme in Computing Sciences
May 2024

This thesis explores the difficulties in lossless compression of data from event cameras. Event cameras have been introduced in the electronic world recently. They have a unique way of capturing data which is rather different from the traditional camera. The cameras capture data at high speed and with low power consumption. The high-speed data capture raises the issue of data storage and transfer. The methods proposed in this thesis take advantage of the temporal and spatial redundancies. The methods are evaluated by comparing the results with existing techniques and datasets. The results achieved depict the best way to compress the event data and their effectiveness compared to the existing techniques.

Keywords: Dynamic Vision Sensor, Neuromorphic camera, Lossless Compression, Event Camera.

The originality of this thesis has been checked using the Turnitin Originality Check service.

PREFACE

I would like to express my profound gratitude to my supervisor, Professor Dr. Ioan Tabus, for all his steadfast guidance, unwavering support, and feedback throughout the process of this thesis. His assistance and encouragement were indispensable, and this thesis could not have been possible.

I would like to extend my gratitude towards my parents for their unwavering support in the entire course of my master's studies. I would also like to thank my brother, who was my constant support and motivation during this time.

I extend my heartfelt thanks to my friends and colleagues for their valuable insights and support, which have been crucial to my thesis.

I would like to express my gratitude for all your support and encouragement in achieving this crucial milestone in my academic journey.

Tampere, 20 May 2024

Dalal Manasi Sameer

CONTENTS

1.INTRODUCTION.....	1
2.BACKGROUND	2
2.1 Dynamic Vision Sensor Event Camera	2
2.1.1 Advantages of DVS.....	5
2.1.2 Applications of DVS	6
2.2 Datasets	7
2.3 Strategies Relevant to Event Data Compression	9
2.3.1 Encoders for One-Dimensional Data.....	9
2.3.1.1. Lempel-Ziv-Markov Chain Algorithm.....	9
2.3.1.2. Prediction by Partial Matching Algorithm	9
2.3.1.3. Burrows-Wheeler Encoding Algorithm	10
2.3.2 Specialized Encoders for Event Data.....	10
2.3.2.1. Spike Coding.....	10
2.3.2.2. Using Point Cloud Compression for Event Data	12
2.4 State-of-the-Art Encoders for Event Data.....	14
3.PROPOSED ALGORITHMS	18
3.1 Summary of the Event Data Used.....	18
3.2 Compressing the Data using 1-Dimensional Encoders	18
3.3 Lossless Compression of DAVIS Camera Event Data using PPMD Encoded Time Differences and GPCC Encoder	30
4.CONCLUSIONS.....	41
REFERENCES.....	43

LIST OF FIGURES

Figure 1.	a. Circuit diagram for the Dynamic Vision Sensor. b. Layers of human retina. [13].....	3
Figure 2.	AER Data Representation	4
Figure 3.	Images captured with (right) and without (left) using a neuromorphic vision-enabled mobile camera. Research carried out by Prophesee in collaboration with Qualcomm [9].....	5
Figure 4.	Images captured by the traditional camera in DAVIS from the dataset introduced by Mueggler et. al., in 2017 in [1]	8
Figure 5.	A Spike-based coding framework [2].....	10
Figure 6.	Location Histogram Map [2]	11
Figure 7.	Motion Vector of Spikes that are encoded in TP mode [2]......	11
Figure 8.	GPCC coding framework used by Khan et. al., in [7].....	12
Figure 9.	Octree Analysis [22].....	13
Figure 10.	Event data extraction from AER data.	14
Figure 11.	Comparison bar plot from previous work.	17
Figure 12.	Various sequencing formats of Data for compression.	19
Figure 13.	Comparison of Compression Ratio from previous work and results achieved	30
Figure 14.	Lossless compression of DAVIS camera event data using PPMD encoded time differences and GPCC encoder.	31
Figure 15.	Comparison of Compression Ratio achieved using 1D encoders and 3D encoders in visual format.	38
Figure 16.	Comparison of compression Speeds from previously achieved results and results achieved by proposed methods.	39
Figure 17.	Comparison of Decompression Speeds from previously achieved results and results achieved by proposed methods.	40

LIST OF TABLES

Table 1.	<i>Neuromorphic cameras available by different brands, their specifications, and applications adapted from W. Shariff et. al. [14]</i>	7
Table 2.	<i>Coding Parameters selected by Martini et al., 2022 [7]</i>	14
Table 3.	<i>Various Compression techniques reimplemented by Khan et. al., in [4] for the comparison of the DAVIS dataset.</i>	15
Table 4.	<i>Comparison of results achieved using various compression techniques by Khan et. al., in [4]</i>	16
Table 5.	<i>Comparison of results achieved using various compression techniques by Khan et. al., in [4] and Martini et al., in [7].</i>	17
Table 6.	<i>Dataset Information.</i>	18
Table 7.	<i>Compression results using LZMA on all components separately and without separating polarities. The components x, y, and p are encoded in their own sequence, while for encoding the element t, the strategy mentioned in the table's header is utilized.</i>	19
Table 8.	<i>Compression results using PPMD on all components separately and without separating polarities. The components x, y, and p are encoded in their own sequence, while for encoding the element t, the strategy mentioned in the table's header is utilized.</i>	20
Table 9.	<i>Compression results using BZip2 on all components separately and without separating polarities. The components x, y, and p are encoded in their own sequence, while for encoding the element t, the strategy mentioned in the table's header is utilized.</i>	20
Table 10.	<i>Compression results using LZMA on all components separately and separated polarities. The components x and y are encoded in their own sequence, while for encoding the element t, the strategy mentioned in the table's header is utilized after separating polarity 0 and polarity 1 in separated sequences.</i>	21
Table 11.	<i>Compression results using PPMD on all components separately and separated polarities. The components x and y are encoded in their own sequence, while for encoding the element t, the strategy mentioned in the table's header is utilized after separating polarity 0 and polarity 1 in separated sequences.</i>	22
Table 12.	<i>Compression results using BZip2 on all components separately and separated polarities. The components x and y are encoded in their own sequence, while for encoding the element t, the strategy mentioned in the table's header is utilized after separating polarity 0 and polarity 1 in separated sequences.</i>	22
Table 13.	<i>Bits per event required to compress x.</i>	23
Table 14.	<i>Bits per event required to compress y.</i>	24
Table 15.	<i>Bits per event required to compress p.</i>	24
Table 16.	<i>Bits per event required to compress t.</i>	25
Table 17.	<i>Bits per event required to compress x with polarities separated.</i>	26
Table 18.	<i>Bits per event required to compress y with polarities separated.</i>	26
Table 19.	<i>Bits per event required to compress t with polarities separated.</i>	27
Table 20.	<i>Compression Ratios obtained by compressing each component separately and using different compression techniques without separating polarities.</i>	28
Table 21.	<i>Compression Ratios obtained by compressing each component separately and using different compression techniques after separating polarities.</i>	28
Table 22.	<i>Comparison of Compression Ratio from previous work and results achieved in this experiment.</i>	29

Table 23.	<i>Configurations used for the GPCC encoder. On the left is the configuration that we adapted from [7] while on the right is the configuration that we found the most efficient with the current version of GPCC codec [22].</i>	33
Table 24.	<i>Permutations used for assigning the components to the axes.</i>	33
Table 25.	<i>Compression ratio achieved using the configuration method1 adapted from Martini et. al. in [7].</i>	34
Table 26.	<i>Comparison of results achieved in this thesis with GPCC codec [22] and results achieved by Martini et. al., in [7].</i>	34
Table 27.	<i>Compression ratios achieved using the 2 configurations and 4 permutations of the axes.</i>	35
Table 28.	<i>Comparison of number of events compressed using configuration taken from Martini et. al. In [7], and PPMD + GPCC.</i>	36
Table 29.	<i>Comparison of results achieved using the proposed method, compressing the partial results with configuration same as Martini et. al., and results achieved by Martini et. al., in [7].</i>	36
Table 30.	<i>Comparison of Compression ratio and Compression + Decompression Times of 1D and 3D encoders</i>	37
Table 31.	<i>Comparison of compression ratio of previously achieved results and results achieved by proposed methods.</i>	37
Table 32.	<i>Comparison of compression and decompression speeds of previously achieved results and results achieved by proposed methods.</i>	39
Table 33.	<i>Comparison of compression ratios and decompression speeds of the 3 proposed methods.</i>	42

LIST OF SYMBOLS AND ABBREVIATIONS

1D	One Dimensional
3D	Three Dimensional
AER	Address Event Representation
AP	Address Prior
APS	Active Pixel Sensor
BWT	Burrows-Wheeler Transform
DAVIS	Dynamic and Active Pixel Vision Sensor
DCM	Direct Coding Method
DVS	Dynamic Vision Sensor
GPCC	Geometry-based Point Cloud Coding
IoT	Internet of Things
LZ77	Lempel-Ziv 1977
LZMA	Lempel-Ziv-Markov Chain Algorithm
MPEG	Moving Pictures Expert Group
MTF	Move to Front
PPMD	Prediction by Partial Matching
RLE	Run Length Encoding
TP	Time Prior
P_B	<i>P component compressed using BZip2 without separating polarities.</i>
P_L	<i>P component compressed using LZMA without separating polarities.</i>
P_P	<i>P component compressed using PPMD without separating polarities.</i>
T_{D4B}	<i>Consecutive time differences compressed after separating in 4 bytes using BZip2 without separating polarities.</i>
T_{D4L}	<i>Consecutive time differences compressed after separating in 4 bytes using LZMA without separating polarities.</i>
T_{D4P}	<i>Consecutive time differences compressed after separating in 4 bytes using PPMD without separating polarities.</i>
T_{DB}	<i>Consecutive time differences compressed using BZip2 without separating polarities.</i>
T_{DL}	<i>Consecutive time differences compressed using LZMA without separating polarities.</i>
T_{DP}	<i>Consecutive time differences compressed using PPMD without separating polarities.</i>
$T_{P_{D4B}}$	<i>Consecutive time differences compressed after separating in 4 bytes using BZip2 after separating polarities.</i>
$T_{P_{D4L}}$	<i>Consecutive time differences compressed after separating in 4 bytes using LZMA after separating polarities.</i>
$T_{P_{D4P}}$	<i>Consecutive time differences compressed after separating in 4 bytes using PPMD after separating polarities.</i>
$T_{P_{DB}}$	<i>Consecutive time differences compressed using BZip2 after separating polarities.</i>
$T_{P_{DL}}$	<i>Consecutive time differences compressed using LZMA after separating polarities.</i>
$T_{P_{DP}}$	<i>Consecutive time differences compressed using PPMD after separating polarities.</i>
X_{P_B}	<i>X component compressed using BZip2 after separating polarities.</i>
X_{P_L}	<i>X component compressed using LZMA after separating polarities.</i>
X_{P_P}	<i>X component compressed using PPMD after separating polarities.</i>
X_B	<i>X component compressed using BZip2 without separating polarities.</i>
X_L	<i>X component compressed using LZMA without separating polarities.</i>

X_P	<i>X component compressed using PPMD without separating polarities.</i>
Y_{P_B}	<i>Y component compressed using BZip2 after separating polarities.</i>
Y_{P_L}	<i>Y component compressed using LZMA after separating polarities.</i>
Y_{P_P}	<i>Y component compressed using PPMD after separating polarities.</i>
Y_B	<i>Y component compressed using BZip2 without separating polarities.</i>
Y_L	<i>Y component compressed using LZMA without separating polarities.</i>
Y_P	<i>Y component compressed using PPMD without separating polarities.</i>

1. INTRODUCTION

Dynamic Vision Sensors (DVS) have gained a lot of popularity in recent times, due to high-speed data capture, high temporal resolution, wide dynamic range, low power requirements, and low bandwidth [1] [2]. Because of these advantages, DVS is useful in high-motion photography, driverless vehicles, robotics, and surveillance. As a result of high-speed data-capturing abilities, challenges like data storage and transmission arise. The data captured by an event camera is different from a traditional camera as it records a stream of events that record the luminance change at each given pixel at a given time.

This thesis aims to tackle this challenge by reducing the data size without losing the high resolution and ability of the event camera to work in real time. This thesis will compare compression results of existing compression techniques and combinations of existing techniques.

This thesis aims to attain high compression without / minimum loss in the data quality by attaining minimum reconstruction error. The results achieved will be compared to similar results previously achieved.

Chapter 2 proceeds to explain the event cameras and DVS, as well as the compression techniques considered during the experiments

Chapter 3 explains the algorithms and the achieved results.

Chapter 4 consolidates the achieved results.

2. BACKGROUND

2.1 Dynamic Vision Sensor Event Camera

The event camera or the Dynamic Vision Sensor (DVS) is a vision sensor that operates in a slightly different way than a traditional frame-based camera. Contrary to conventional cameras that capture images (frames) at a fixed interval, an event camera captures “events” when it detects a change in luminance at every pixel asynchronously. When there is a change in luminance, the sensor creates an event that records the time stamp t , pixel location x and y , and polarity p which is either 0 or 1 depending on if the brightness at that pixel has increased (1) or decreased (0). Event cameras work on the same principle as any living organism in terms of processing the visuals in a sense where they only notice change in terms of brightness changes. Since the event camera only captures brightness change, the data can be transmitted with speed in order of microseconds, it has a high dynamic range (up to 80dB more than a traditional camera). Some of the other features of an event camera are low storage, and low power requirements [1]. Due to these advantages, they can be used in a wide range of applications from driverless cars to reducing motion blur.

Frame-based image sensors suffer from high power consumption and high processing time. A DVS was developed with a bio-inspired model having a three-layer retina. This model was implemented with a simple version of the photoreceptor-bipolar-ganglion pathway [10]. Traditional cameras record data irrespective of the change at each pixel which increases the data rate and volume for no apparent reason. A biological vision system works on the principle of capturing data based on the changes in the scene [10]. In a human retina, the photoreceptor layer is made up of light-sensitive cells, which are responsible for the conversion of incoming light signals into electric signals, which then drive the bipolar cells. The two types of bipolar cells- The On-bipolar cell and the Off-bipolar cell are in charge of the coding of bright and dark luminance changes. The bipolar cells then trigger the ganglion cells, The ganglion cells carry the data to the visual cortex [13]. Figure 1 shows the circuit for the Dynamic Vision Sensor and layers of the human retina as explained by Chen et al., 2020 in [13]. Figure 1b shows the layers of the human retina. Figure 1a shows the circuit for DVS and the layers corresponding to the human retina are marked in coloured dotted boxes. Silicon retinas consist of a photoreceptor and chip having a 2D hexagonal grid of pixels, which replicate the human retina. The representation of Silicon retinas had the major contribution of Tobi Delbruck and Christoph Posch. When the silicon retinas were developed by them, they faced a major challenge, where

each pixel needed its own wire. To overcome this issue, Address Even Representation was introduced by Caltech group of Carver Mead [13].

As depicted in Figure 1a DVS pixel copies the human retina layers for the transmission of information from the photoreceptors. The pixels work separately and add important spatial and temporal information to the luminance change. They trigger events when the threshold is crossed in either direction. The working principle of DVS varies from the human retina, except the key features like the luminance change, event data and different output channels form On and OFF events are maintained [13].

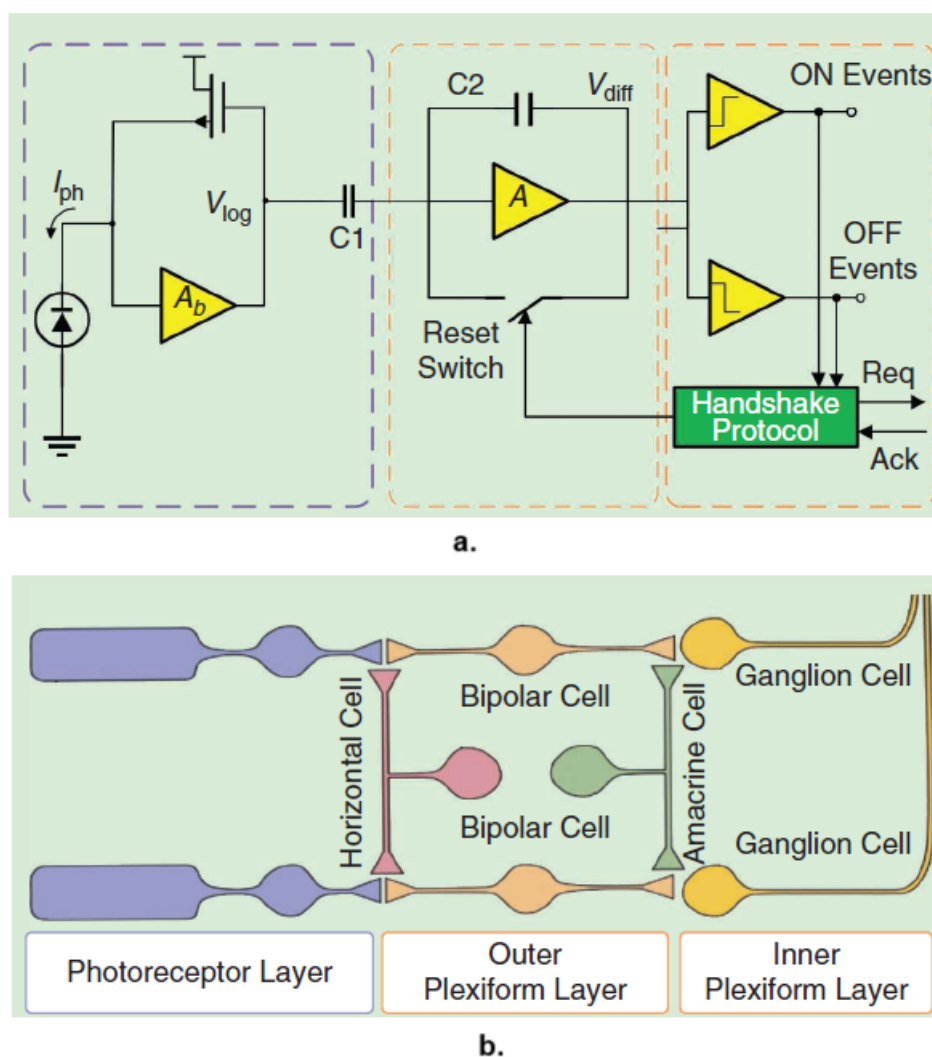


Figure 1. a. Circuit diagram for the Dynamic Vision Sensor. b. Layers of human retina. [13]

The DVS reports asynchronous log-intensity changes and discards the absolute information changes. This information is essential for object detection and classification [11]. To overcome this issue a Dynamic and Active Pixel Vision Sensor (DAVIS) was introduced by Brandli et. al., in [11]. The DVS and Active pixel sensor (APS) share a single photodiode. They have a

resolution of 240x180 pixels, fabricated on a 0.18 μ m 6M1P CMOS image sensor (CIS). It has a dynamic range of 130 dB.

Dynamic and Active pixel Vision Sensor (DAVIS) is a combination of traditional and vision sensor cameras in a way that transmit events in addition to frames. The resolution of events is in microseconds. Each event consists of a tuple $\langle x, y, t, p \rangle$, where x and y are the pixel coordinates, t is the time stamp and p is polarity with a value of either 0 or 1 of the events. The DAVIS camera output consists of the visual data that is the events data, the angular and linear motion of the sensor and the intensity of the images. This data representation is known as the Address Event Representation (AER). In AER data representation, each row is an event. The size of each event is 64 bits. Figure 2 shows the AER representation extracted from [4]. It is a series of 8-bit words. 1 bit is reserved for vision sensor type. The X address is represented in 10 bits, while the Y address is represented in 9 bits. The polarity flag is represented using 1 bit and the DVS trigger information is represented using 1 bit. “The Asynchronous Time-based Image Sensor (ATIS)” has acceleration, Gyroscope and temperature sensors built in. The data from these sensors are represented in 10 bits reserved for Analog to Digital Converter (ADC) of this data. The time stamp information is represented in 32 bits [4].

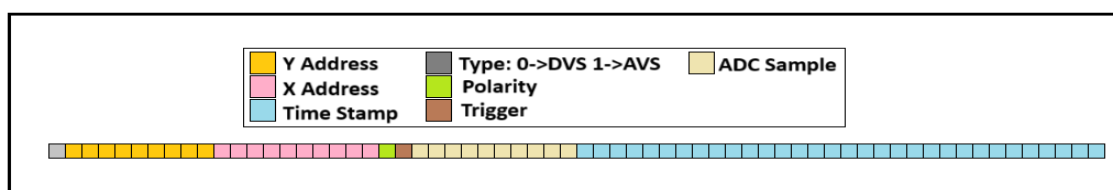


Figure 2. AER Data Representation

Prophesee has been working with Qualcomm and Sony to create a neuromorphic vision system. They are using their patented sensor design and AI algorithms to recreate a vision similar to a human. Prophesee in collaboration with Qualcomm has introduced a neuromorphic-enabled vision to mobile cameras, which can capture images and videos with extremely low motion blur [9]. Figure 3 [9] shows the side-by-side comparison of images captured with and without event cameras. The image using the neuromorphic vision-enabled mobile camera (right) was able to reduce the blur that occurred due to movement. The image was extracted from [9].



Figure 3. Images captured with (right) and without (left) using a neuromorphic vision-enabled mobile camera. Research carried out by Prophesee in collaboration with Qualcomm [9]

2.1.1 Advantages of DVS

Event cameras overpower the traditional cameras with the help of benefits like:

- **High temporal resolution:** Event cameras capture data as a stream of events, which results in a temporal resolution in the order of microseconds. They can detect rapid changes in the scenes which makes them suitable for applications requiring low latency.
- **Low power consumption:** Since the event cameras are activated only when there is a change in brightness as opposed to the traditional camera that captures a fixed rate of frames per second, the event camera consumes less power (around 10mW) than a traditional camera.
- **Low motion blur:** Since the camera detects changes in brightness, it is more robust to motion blur as compared to the traditional camera. In addition to high temporal resolution, the ability to record brightness changes at each pixel, allows the DVS to track the movement precisely.
- **High Dynamic Range:** DVS can trigger and record events in low as well as high brightness scenes without losing any details of the scene. The dynamic range of a DVS is as high as 130 dB.

2.1.2 Applications of DVS

The Dynamic Vision Sensor has a wide range of applications:

- **Surveillance:** The ability to detect rapid changes, makes the DVS ideal for security and surveillance applications at locations like malls, airports, banks to private properties.
- **Autonomous cars:** The ability to detect changes quickly, and to detect them in low or high brightness, enables DVS to contribute to precise navigation and control, which is ideal for driverless cars.
- **Robotics:** The ability to provide real-time data, due to its low latency opens a wide range of possibilities in robotics like, object detection, object tracking and navigation.
- **High Motion Photography:** Due to low latency and low motion blur, DVS is ideal for high motion photography.
- **Deblurring:** DVS captures events whenever there is a brightness change in the scene. This enables DVS to track a scene very precisely, with the use of machine learning algorithms, the motion can be deblurred using the event data from DVS.

Table 1 is adapted from W. Shariff et. al. [14], it consolidates the various brands working on the neuromorphic cameras with the specifications and applications these cameras are used in. Companies like Prophesee, iniVation, Century Arks, CelePixels, Samsung, and Insightness are working on neuromorphic camera technology. The dynamic range of these cameras varies from 55dB to 120 dB. These cameras have a resolution range from 320 x 240 pixels up to 1280 x 960 pixels. The dynamic range shows the ability of these cameras to capture events in very dark and very bright scenes. The latency of these cameras show how fast the camera can detect the changes with figures as low as 10 μ s.

Brand	Model	Resolution	Latency (μ s)	Dynamic Range (dB)	Key features	Applications
Prophesee [15]	GENX320, IMX636 (SONY), IMX646 (SONY)	320 x 320 - 1280 x 720	< 100	120+	Anti-flicker filtering, Event rate Controller, High temporal resolution, Low power consumption	AV / VR / XR, Healthcare (privacy) cameras, Wearables, Smart home, Industrial, Automotive, IoT
iniVation [16]	DAVIS346, DAVIS346-AER, DVXplorer-S-Duo, DVXplorer-Micro-Lite	320 x 240, 346 x 260, 640 x 480	100	55 - 120	High temporal resolution, Energy efficiency	Robotics, Surveillance, Research
Century Arks [17]	SilkyEvCam	640 x 480 - 1280 x 720	100 – 200	120+	High-speed, High-resolution	Industrial (piece counting, vibration detection, etc), Security, Robotics, Chemistry, Research
CelePixel [18]	CeleX-IV, CeleX-V	768 x 640 - 1280 x 800	10	120+	High-speed, High-resolution	AR, VR, ADAS
Samsung [19]	DVS Gen1, Gen2, Gen3, Gen4	640 x 480 - 1280 x 800	50 – 150	90-100	High efficiency, Event-based processing	Consumer electronics, Automotive
Insightness [20]	Rino-3, Rino-4	320 x 262, 800 x 600, 640 x 480	< 100	100+	Silicon retina technology, Versatile lighting adaptation	Robotics, Drones, Wearables

Table 1. *Neuromorphic cameras available by different brands, their specifications, and applications adapted from W. Shariff et. al. [14]*

2.2 Datasets

This thesis uses a subset of data presented by Mueggler et al., 2017 [1]. The datasets contain different scenes, increasing speeds, and multiple degrees of freedom. According to Khan et al., 2020 [4], this dataset is designed for high dynamic and high-speed computer vision and robotics applications for both outdoor and indoor scenarios.

The DAVIS dataset frames as viewed from a traditional camera are shown in Figure 4. The maximum value for the x component is 240 and that for the y component is 180 [4].

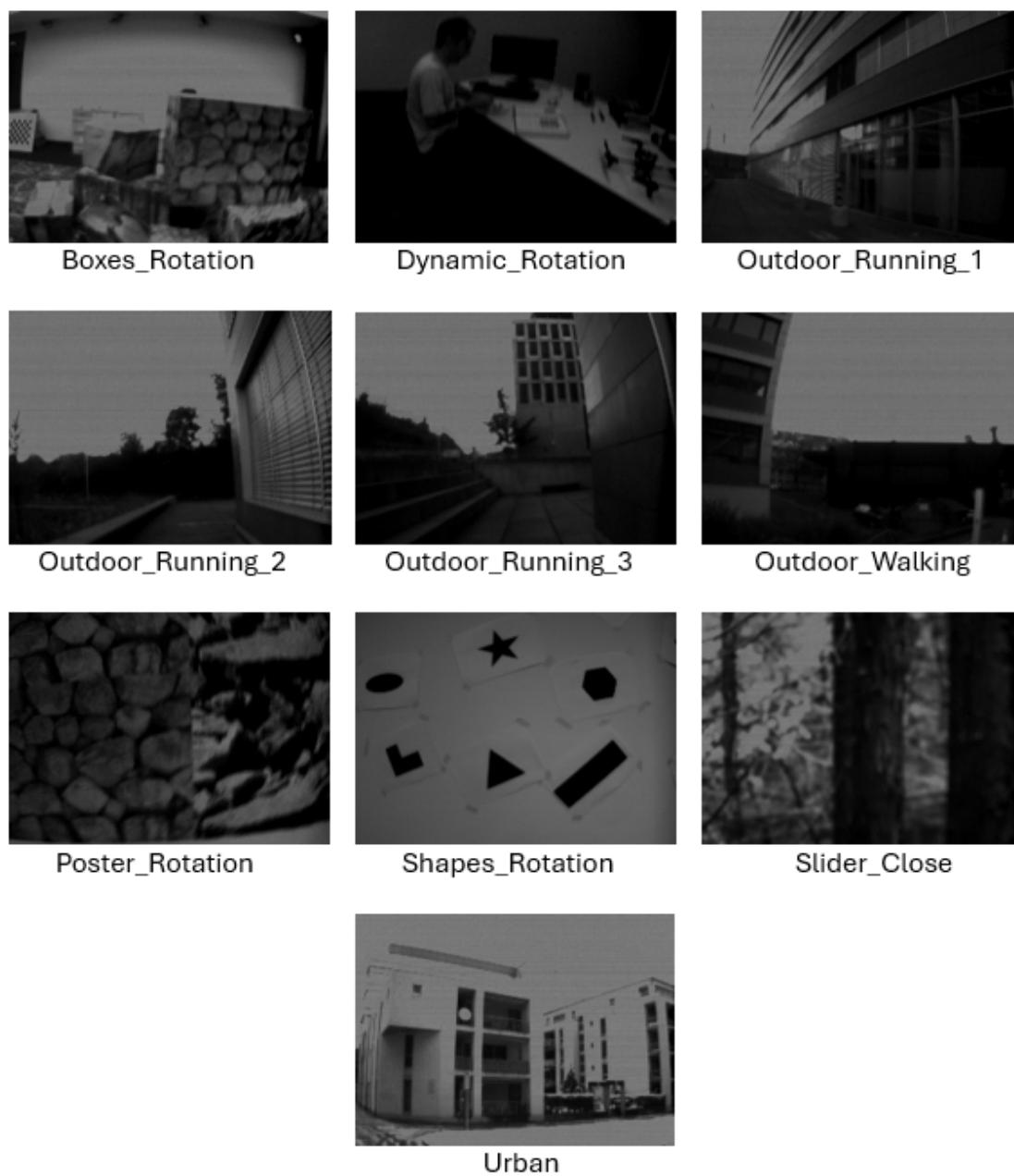


Figure 4. Images captured by the traditional camera in DAVIS from the dataset introduced by Mueggler et. al., in 2017 in [1]

2.3 Strategies Relevant to Event Data Compression

2.3.1 Encoders for One-Dimensional Data

2.3.1.1. Lempel-Ziv-Markov Chain Algorithm

Lempel-Ziv-Markov Chain Algorithm (LZMA) is part of the family of LZ77 (Lempel-Ziv 1977) compression algorithms. It was used to optimize LZ77 by getting faster compression and decompression in addition to a higher compression rate and lower memory requirements [5]. It compresses files using a combination of dictionary-based encoding and statistical modelling. Statistical modelling enables the analysis of entire data, while the dictionary-based encoder compresses small data simultaneously.

Y. Horita et al., 2019 explains the key steps in the algorithm as follows [5]:

1. Delta encoding: This step involves data filtering that encodes the input stream to produce an output where each word (8 bytes) represents the difference between the current word and the previous word. The initial word remains unchanged as the first word in the sequence.
2. Sliding dictionary encoding: This step is an extension of LZ77 designed to accommodate larger dictionaries. The resulting output sequence includes the distance from a string found in the look-ahead buffer, the length of the matched string, and the new symbol from the input.
3. Range encoding: Instead of using arithmetic coding, LZMA uses range encoding as it gives better compression efficiency. This step adapts based on the context. This step uses the output from step 2 as its input.

2.3.1.2. Prediction by Partial Matching Algorithm

Prediction by Partial Matching (PPM) algorithm uses context modelling. It is not as widely used as LZMA, but this algorithm is considered especially good for text and data with patterns that can be predicted. It predicts the next symbol in the input stream based on previously encoded symbols. This algorithm keeps an adaptive probability prediction for encoding symbols based on the previous data. It updates the probabilities dynamically and adjusts the model as the data stream is being processed. The PPM is one of the most efficient versions of the PPM algorithm. It considers variable length streams to predict the next symbol. It keeps a dictionary of previous strings with their probabilities to ensure accurate predictions and encoding symbols. It works by creating a model of the previously received input stream. This model is based on the probability distribution of the input data and the context of the previous symbols. This context can be a single symbol or multiple symbols. This method is computationally intensive, which makes it less suitable for devices with fewer resources and limitations on power consumption.

2.3.1.3. Burrows-Wheeler Encoding Algorithm

BZip2 follows the Burrows-Wheeler Transform (BWT) and Run-Length Encoding (RLE). It is mostly used for the compression of large files. It applies selective BWT to arrange the data in a more compressible way by rearranging the input stream in such a way that shows repetitive patterns. The algorithm uses the concept of Move to Front (MTF) algorithm by putting symbols with higher probability at the start of the data stream. BZip2 uses RLE for the compression of this sequence with repeated symbols. Contrary to storing each repeated symbol separately, BZip2 stores each symbol and the number of repetitions to decrease redundancy. Post BWT and RLE, BZip2 uses entropy encoding like Huffman encoding to assign shorter codes to the frequently occurring strings. Dividing the data into smaller blocks and then applying BWT and RLE helps in managing the large data. It also improves the coding efficiency of entropy encoders like Huffman.

2.3.2 Specialized Encoders for Event Data

2.3.2.1. Spike Coding

Moving to DVS-specific encoding methods, “A cube-based spike coding framework” was proposed by Bi et al., 2018 in [2]. To take advantage of the spatial and temporal redundancy in DAVIS data a cube-based spike coding framework was proposed as shown in Figure 5 [2].

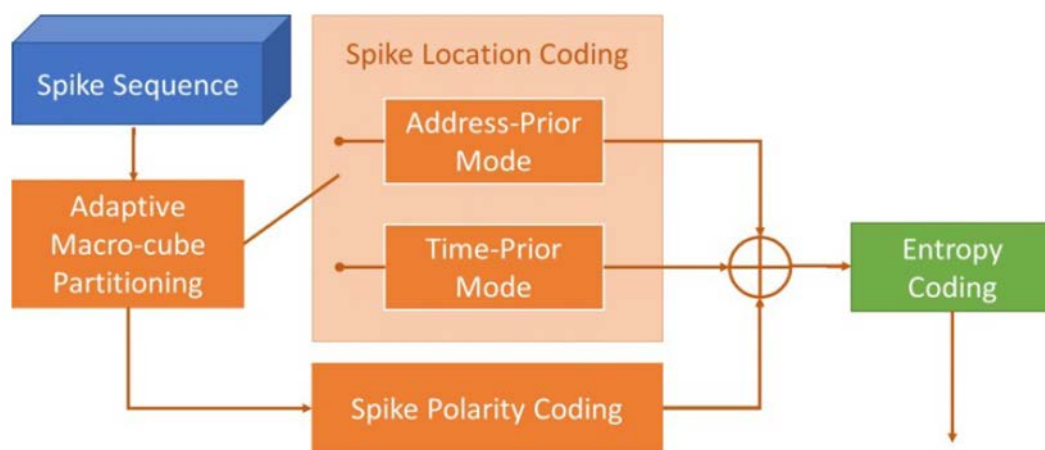


Figure 5. A Spike-based coding framework [2]

The spike sequence obtained from the DVS data can be partitioned into macro cubes temporally which have the complete spatial data. This macro cube is then spatially split into smaller spike cubes.

The encoding takes place in two steps:

1. Spike Location Coding:

Spike distribution is classified in two ways – spatial centralized and spatial decentralized. Spikes generated due to object movement are classified as spatial centralized distribution and spikes fired globally duty luminance change of the scene are classified as spatial decentralized coding [2]. To address this, two types of address modes are defined as follows:

i. Address Prior Mode:

In Address Prior Mode (AP Mode), all the spikes are projected on an x-y plane in a location histogram. Location histogram is represented by having a location histogram map and the counts. The histogram map is a binary depiction showing if that pixel location has a count or not and the histogram map count shows the number of spikes at that location [2]. The same is shown in Figure 6.

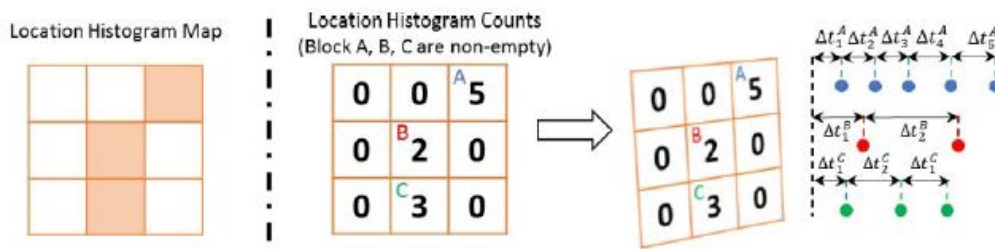


Figure 6. Location Histogram Map [2]

ii. Time Prior Mode:

In Time Prior Mode (TP Mode), a centre point (x_c^*, y_c^*) and all the other spikes are projected to this time axis. The centre point is calculated by using the formula [2]:

$$x_c^*, y_c^* = \arg \min_{x_c, y_c} |x_i - x_c| + |y_i - y_c|$$

where (x_i, y_i) is the spike address in the spatial. The centre point (x_c^*, y_c^*) is encoded either directly or differentially by referring to the centre point (x_c, y_c) of the previous spike cube. Since the spikes are spatially centralized, the spatial offset locations (x_i, y_i) form a motion vector which is then encoded as depicted in Figure 7 [2].

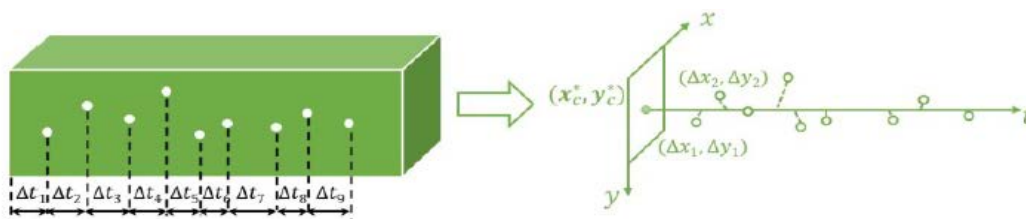


Figure 7. Motion Vector of Spikes that are encoded in TP mode [2].

2. Spike Polarity Coding

The second step for Spike coding is the Spike polarity coding. As mentioned earlier, when the polarity of the spike is “ON” or 1 there is an increase in luminance and “OFF” or 0 means a decrease. The temporal correlation is high when two consecutive event pair polarities are analysed. Encoding of spike polarity takes advantage of this relation.

2.3.2.2. Using Point Cloud Compression for Event Data

Point cloud compression reduces the size of a 3D point cloud without loss in the geometric and attribute information. The point cloud describes a surface or an object in 3D space. It exploits the temporal and spatial correlation between the neighbouring points. It was standardized by the MPEG to ensure effective encoding practices for point cloud data using geometry-based and attribute-based encoding.

According to Huang and Ebrahimi et al., 2023 [6], the basic idea is to use a PCC compressor to compress events as a point cloud by considering time stamp t as the third axis with coordinates (x, y, t) . The Geometry Point Cloud Coding (G-PCC) encodes the point cloud using an octree structure. It mainly focuses on irregular point clouds [6].

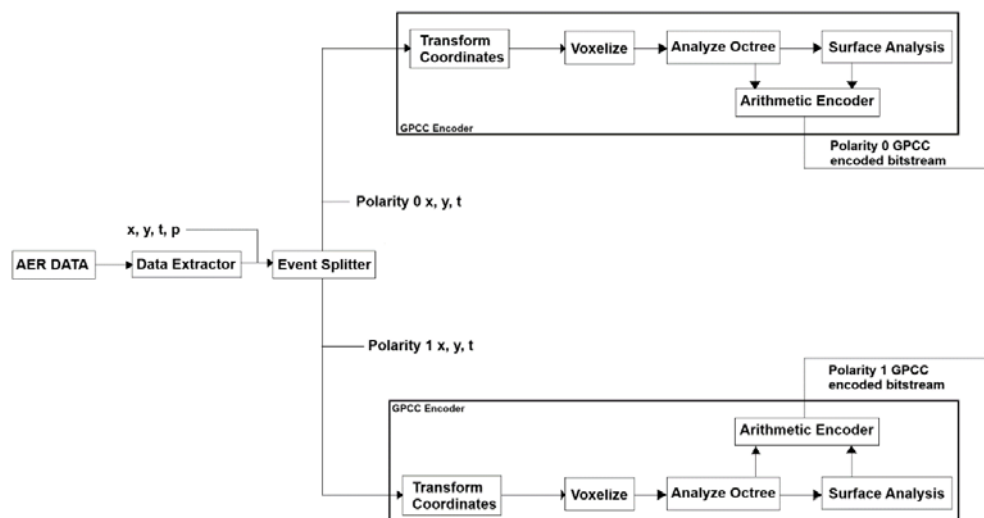


Figure 8. GPCC coding framework used by Khan et. al., in [7]

Figure 8 shows the GPCC dataflow as implemented by Martini et. al., in [7]. The first step in Geometry-based Point Cloud Coding (G-PCC) is to perform coordinate transformation followed by voxelization [7]. In voxelization, the cloud points are initially quantized. After quantization, there is a possibility that the cloud points can be duplicated. Points with the same position and different attributes are then merged into a single point. This process of quantizing and then grouping of points is known as voxelization. The group of points with the same position and different attributes is known as a voxel. Geometry analysis is then done by using the

Position related parameters	
mode	0
mergeDuplicatedPoints	0
srcResolution	0
outputResolution	0
positionQuantizationScale	1
trisoupNodeSizeLog2	0
neighbourAvailBoundaryLog2	8
intra_pred_max_node_size_log2	6
inferredDirectCodingMode	1
maxNumQtBtBeforeOt	4
minQtbtSizeLog2	0
planarEnabled	1
planarModeIdcmUse	0
convertPlyColourSpace	1
transformType	0
numberOfNearestNeighborsInPrediction	3
levelOfDetailCount	12
intraLodPredictionSkipLayers	0
interComponentPredictionEnabled	0
positionBaseQp	0
sliceMaxPoints	107000000
enforceLevelLimits	0

Table 2. Coding Parameters selected by Martini et al., 2022 [7]

Table 2 shows the coding parameters used for their experiment.

2.4 State-of-the-Art Encoders for Event Data

Khan et al., 2020 in [4] extracted the relevant data from the AER data, in which the time stamp with 32 bits was divided into 4 integers of 8 bits each. The extracted data now had 7 columns out of which 4 columns were of the time stamp and the remaining 3 columns were of spatial address x and y and polarity p. Further they organized these 7 columns in two formats the row major and column major format. This process is shown in Figure 10.

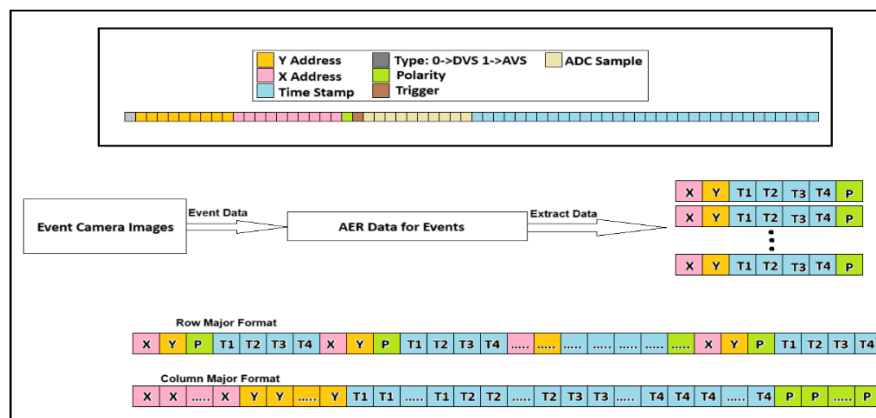


Figure 10. Event data extraction from AER data.

In the row-major format, the events were placed consecutively, in the memory. Whereas, in the column-major format, the same elements of each event were placed consecutively. Additionally, in column-major format, the timestamp stored in 32 bits is divided into 4 words taking up 8 bits each. Each word for all the events is then placed consecutively.

The row-major format was primarily used for the IoT-specific compression techniques, while the column-major format was used for the compression of Dictionary, Entropy and Fast Integer compression techniques.

In previous studies, compression techniques were compared based on various parameters like compression ratio, compression speed, and decompression speed [4]. For the scope of this thesis, we will be focussing only on compression ratio.

The compression ratio is calculated using the formula [4]:

$$CR = \frac{N_{events} \times 8}{\gamma}$$

Where γ is the size in bytes of the compressed stream, and N_{events} the number of spike events [4].

Entropy Coding	Huffman Coding	Fast Integer Compression	Simple 8B
	Arithmetic Coding		SIMDBP128
Dictionary-based Compression	Brotli		FastPFOR
	LZMA		Snappy
	LZ4 / LZ77		Memcpy
	Zstb		Sprintz-FIRE
	Zlib	Sprintz-Delta-Huff	
DVS Compression	SPIKE Coding	IoT Specific Compression	Sprintz-Delta

Table 3. Various Compression techniques reimplemented by Khan et. al., in [4] for the comparison of the DAVIS dataset

Khan et. al., compared results from various compression techniques in [4]. Authors of [4] considered entropy coding techniques like Huffman [29] and Arithmetic [30] coding. In dictionary-based encoding, encoders like Brotli [31], Lempel Ziv Markov Chain Algorithm [32], Lempel Ziv 1977 [33] (LZ77 / LZ4), Zstandard (Zstd) [34] and Zlib [35] were used to compare the compression techniques. The authors also compared the DVS-specific compression technique Spike coding [2] for comparison. Fast Integer Compressors like Simple8B [36], SIMDBP128 [37], FastPFOR [37], Snappy [38], and Memcpy which are known for their fast compression speeds were also considered. IoT specific algorithm known as Sprintz developed by Blalock et al., 2018 in [39] was also used to compare the compression ratio. Various versions of Sprintz were considered for the comparison. Sprintz-FIRE combines the predictive coder Fast Integer Regression (FIRE) and the Huffmann encoding, Sprintz-Delta uses the delta encoding in place of FIRE and skips Huffmann encoding. Sprintz-Delta-Huff takes advantage of

delta and Huffman encoding. Khan et. al., reimplemented these techniques in [4] to achieve the compression ratio summarized in Table 4.

The compression techniques reimplemented by Khan et. al. in [4] are summarised in Table 3.

Sequence	Spike Coding	LZMA	Brotli	Zlib	Zstd	LZ4	Spritz Delta-Huf	Spritz FIRE	Spritz Delta	Huffman	Snappy	FastPFOR	SIMDBP128	Simple8B	Memcpy
Boxes_Rotation	4.95	4.92	4.38	4.2	4.1	3	3.83	3.72	2.83	1.96	2.98	1.4	1.38	1.25	1.12
Dynamic_Rotation	3.85	3.34	3.19	3.1	3.1	2.5	2.68	2.63	2.33	1.89	2.44	1.31	1.28	1.15	1.12
Outdoors_Running_1	3.68	3.25	3.09	3.1	3	2.4	2.6	2.58	2.33	1.87	2.37	1.35	1.32	1.18	1.12
Outdoors_Running_2	3.92	3.41	3.26	3.2	3.1	2.5	2.7	2.67	2.35	1.89	2.48	1.3	1.27	1.15	1.12
Outdoors_Running_3	3.97	3.49	3.32	3.3	3.2	2.6	2.75	2.72	2.36	1.9	2.53	1.29	1.26	1.14	1.12
Outdoors_Walking	3.54	3.11	2.89	2.9	2.8	2.2	2.53	2.52	2.3	1.84	2.24	1.35	1.31	1.18	1.12
Poster_Rotation	4.88	4.77	4.26	4.1	4	3	3.7	3.6	2.76	1.96	2.92	1.4	1.37	1.24	1.12
Shapes_Rotation	3.78	3.04	2.78	2.8	2.7	2.2	2.46	2.43	2.26	1.79	2.21	1.34	1.31	1.17	1.12
Slider_Close	3.84	3.19	2.85	2.9	2.8	2.3	2.65	2.62	2.36	1.79	2.26	1.41	1.36	1.23	1.12
Urban	3.45	3.13	2.93	2.9	2.8	2.3	2.58	2.54	2.31	1.83	2.31	1.36	1.35	1.22	1.12
Total Average	3.99	3.57	3.3	3.2	3.2	2.5	2.85	2.8	2.42	1.87	2.47	1.35	1.32	1.19	1.12

Table 4. Comparison of results achieved using various compression techniques by Khan et. al., in [4]

The results achieved by Khan et. al., 2020 depict, to achieve the best compression, event data-specific encoders like Spike Coding, which is a 3D encoder, use the correlation between the inputs to give the best results. 1D encoders like LZMA are second to achieve the next best results, they use the data redundancies to achieve these results.

Table 5 compares the results published by Khan et. al., in [4], and Martini et al., in [7] (The spike encoding method [2] was reimplemented by Khan et. al., in [4]). This table depicts that the 3D encoders perform much better as compared to the 1D encoder. The best results are marked in bold.

The Point Cloud Compression achieves the best results, as it utilizes the 3D redundancies existing in the event data in the most efficient way.

These results are summarized in a visual format in Figure 11.

Sequence	1D Encoder	3D Encoders	
	LZMA (Access 2020) [4]	Spike Coding (DCC 2018) [2]	Point Cloud Compression (ACCESS 2022) [7]
Boxes_Rotation	4.92	4.95	5.35
Dynamic_Rotation	3.34	3.85	4.99
Outdoors_Running_1	3.25	3.68	3.97
Outdoors_Running_2	3.41	3.92	4.46
Outdoors_Running_3	3.49	3.97	4.67
Outdoors_Walking	3.11	3.54	3.84
Poster_Rotation	4.77	4.88	5.44
Shapes_Rotation	3.04	3.78	4.39
Slider_Close	3.19	3.84	3.62
Urban	3.13	3.45	3.66

Table 5. Comparison of results achieved using various compression techniques by Khan et. al., in [4] and Martini et al., in [7].

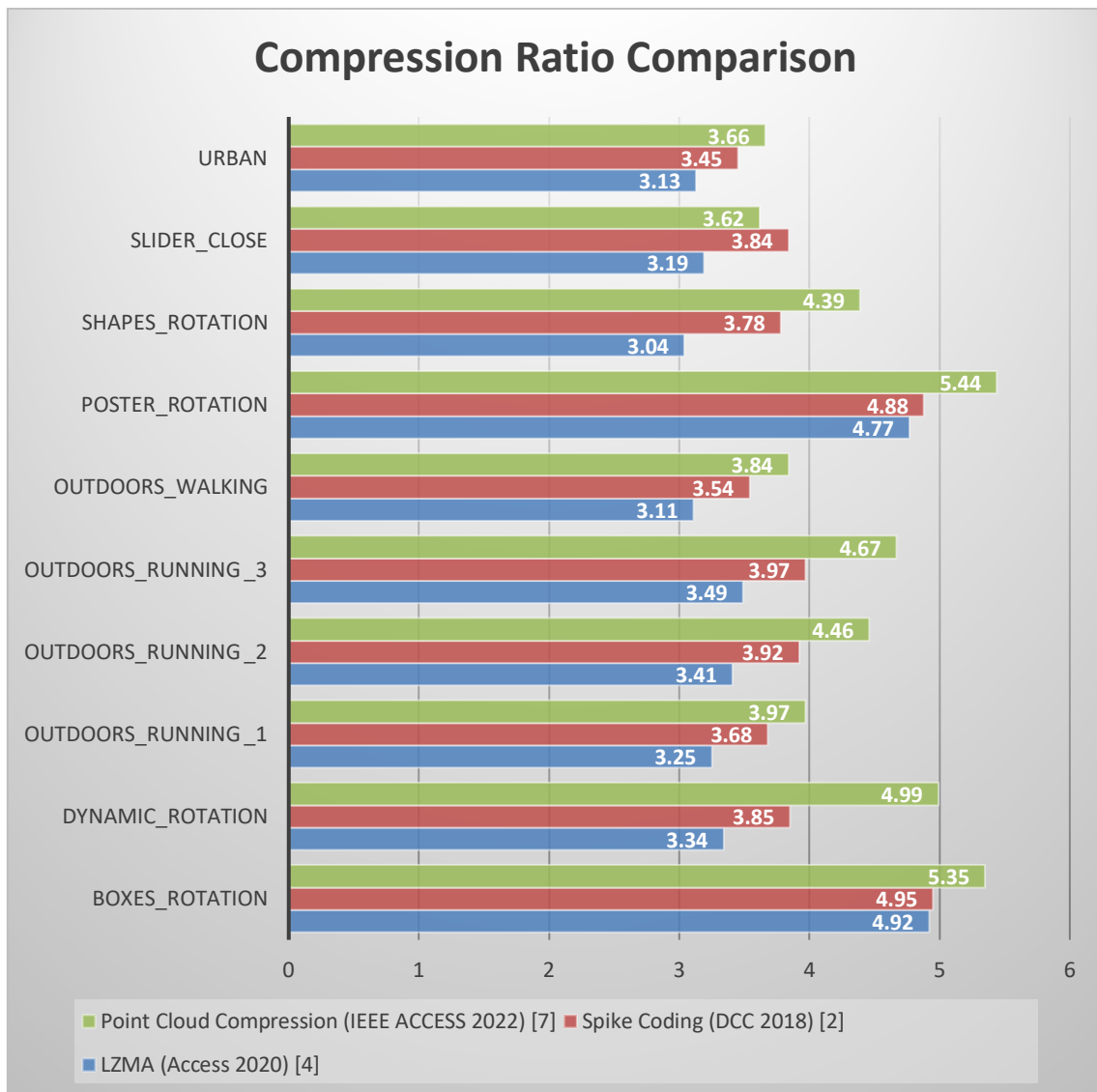


Figure 11. Comparison bar plot from previous work.

3. PROPOSED ALGORITHMS

In this thesis, we plan to compress each component separately using various data compression algorithms as discussed in Section 2.

3.1 Summary of the Event Data Used

In this thesis, we used the sequences Boxes_Rotation, Dynamic_Rotation, Outdoors_Running, Outdoors_Walking, Poster_Rotation, Shapes_Rotation, Slider_Close, and urban from the dataset [1]. The time durations of these datasets are summarized in Table 6. The unit of event rate is kilo events per second.

Sequence	Event Rate (kev/s)	Sequence Duration (s)
Boxes_Rotation	4288.65	5 (45–50)
Poster_Rotation	4021.1	5 (45–50)
Dynamic_Rotation	1077.73	20 (0–20)
Slider_Close	336.78	3(0–3)
Shapes_Rotation	245.61	20 (0–20)
Outdoors_Running_3	1525.5	20 (40–60)
Outdoors_Running_2	1229.4	20 (20–40)
Outdoors_Running_1	713.8	20 (0–20)
Urban	503.04	10 (0–10)
Outdoors_Walking	342.2	20 (0–20)

Table 6. *Dataset Information*

3.2 Compressing the Data using 1-Dimensional Encoders

In the first proposal, we compress each component separately using 3 compression techniques- LZMA, PPMD, and BZip2.

While compressing the t component, 4 methods were used to process t before it was compressed -

1. timestamp (t) – In this method, 32 bits are used to write the time. According to Figure 12, format 1 is used.
2. differential t – In this method, the time stamp of first event t_0 is written using 32 bits and for the remaining time stamps the difference between the current and previous time stamp is written using less than 32 bits. t_0 is compressed separately and differential t is compressed separately. According to Figure 12, format 2 is used after differential t .
3. dividing t in 4 bytes – In this method, all the time stamps are divided into 4 words, Words 1 through 4, Word 1 being the Most Significant Byte and Word 4 being the Least Significant Byte. These 4 Words are then written separately using 8 bits per word and compressed separately. According to Figure 12, format 3 is used.
4. differential t in 4 bytes – This method is a combination of the second and the third method. In this method, the result of method 2, the time stamp of the first event t_0 and

the remaining time stamps in their differential form are then separated into 4 bytes as done in Method 3. Thus, the output of this method, t_0 and the differential t of remaining time stamps separated into 4 words, which are written separately. t_0 is compressed separately and differential t is compressed separately. According to Figure 12, format 4 is used after differential t .

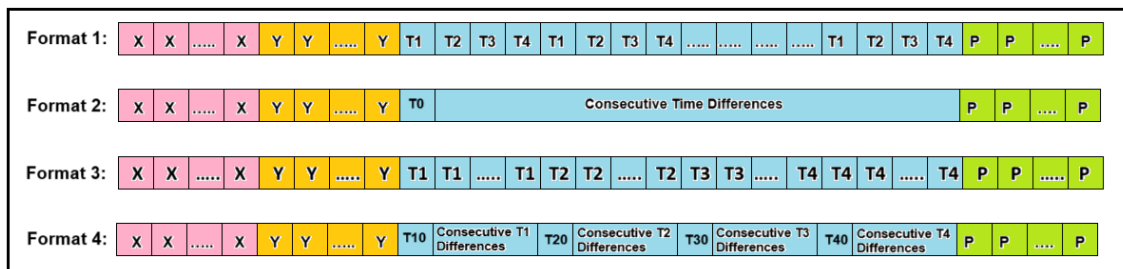


Figure 12. Various sequencing formats of Data for compression.

In the first part of this experiment, all the components were compressed using the same compression technique and the polarities 0 and 1 were not separated.

In Table 7 - Table 9, the various compression techniques for timestamps as discussed earlier are represented using the following nomenclature.

1. For method 1, <CompressionTechnique> (LZMA, PPMD, BZIP2) is used.
2. For method 2, where the consecutive time differences are encoded, <Compression-Technique_Diff_T> (LZMA_DIFF_T, PPMD_DIFF_T, BZIP2_DIFF_T) is used.
3. For method 3, where the time differences are divided into 4 bytes and then encoded, <CompressionTechnique_T_4_bytes> (LZMA_T_4_bytes, PPMD_T_4_bytes, BZIP2_T_4_bytes) is used.
4. For method 4, where the consecutive time differences are separated in 4 bytes and then encoded, <CompressionTechnique_Diff_T_4_bytes > (LZMA_DIFF_T_4_bytes, PPMD_DIFF_T_4_bytes, BZIP2_DIFF_T_4_bytes) is used.

Sequence	LZMA	LZMA_Diff_T	LZMA_T_4_bytes	LZMA_Diff_T_4_bytes
Boxes_Rotation	5.2697	5.4051	5.1734	5.4048
Dynamic_Rotation	3.4651	3.5833	3.4669	3.6094
Outdoors_Running_1	3.602	3.7457	3.602	3.7456
Outdoors_Running_2	3.5274	3.6739	3.5282	3.6738
Outdoors_Running_3	3.3722	3.4978	3.3632	3.4977
Outdoors_Walking	3.2451	3.3831	3.2075	3.3836
Poster_Rotation	5.0728	5.2269	5.0073	5.2266
Shapes_Rotation	3.1667	3.31	3.1219	3.3098
Slider_Close	4.1198	4.2885	4.066	4.2863
Urban	3.2637	3.3903	3.2327	3.3899

Table 7. Compression results using LZMA on all components separately and without separating polarities. The components x , y , and p are encoded in their own sequence, while for encoding the element t , the strategy mentioned in the table's header is utilized.

Table 7 shows the compression results achieved by compressing x, y, p, and t separately using LZMA. The time stamp was compressed using the 4 methods. The best results were achieved when the time was compressed after taking the consecutive differences, either with or without separating the time stamp in 4 bytes that is using Format 2 or Format 4 according to Figure 12. While compressing, the polarities were not separated.

Table 8 shows the compression results achieved by compressing x, y, p, and t separately using PPMD. The time stamp was compressed using the 4 methods. The best results were achieved when the time was compressed after taking the consecutive differences without separating the time stamp in 4 bytes that is, using Format 2 as shown in Figure 12. While compressing the polarities were not separated.

Sequence	PPMD	PPMD_Diff_T	PPMD_T_4_bytes	PPMD_Diff_T_4_bytes
Boxes_Rotation	3.8558	5.4588	5.4193	5.4587
Dynamic_Rotation	2.1642	3.4917	3.4821	3.4989
Outdoors_Running_1	2.2841	3.6534	3.6307	3.6534
Outdoors_Running_2	2.1987	3.5765	3.5557	3.5765
Outdoors_Running_3	2.0326	3.4088	3.3866	3.4088
Outdoors_Walking	1.8927	3.2859	3.2546	3.2858
Poster_Rotation	3.6775	5.2459	5.2094	5.2458
Shapes_Rotation	1.8225	3.1409	3.1051	3.1408
Slider_Close	2.7086	4.346	4.2916	4.3452
Urban	2.0074	3.2896	3.271	3.2895

Table 8. *Compression results using PPMD on all components separately and without separating polarities. The components x, y, and p are encoded in their own sequence, while for encoding the element t, the strategy mentioned in the table's header is utilized.*

Sequence	BZip2	BZip2_Diff_T	BZip2_T_4_bytes	BZip2_Diff_T_4_bytes
Boxes_Rotation	4.494	5.1818	5.1604	5.1817
Dynamic_Rotation	2.3728	3.4267	3.4141	3.4318
Outdoors_Running_1	2.6338	3.5962	3.5767	3.5962
Outdoors_Running_2	2.4699	3.5205	3.5008	3.5205
Outdoors_Running_3	2.1899	3.3558	3.3412	3.3558
Outdoors_Walking	1.9264	3.2204	3.1987	3.2203
Poster_Rotation	4.3122	4.9789	4.9601	4.9788
Shapes_Rotation	1.8468	3.1025	3.0765	3.1023
Slider_Close	2.732	4.1783	4.1231	4.1771
Urban	2.0638	3.2461	3.2218	3.2459

Table 9. *Compression results using BZip2 on all components separately and without separating polarities. The components x, y, and p are encoded in their own sequence, while for encoding the element t, the strategy mentioned in the table's header is utilized.*

Table 9 shows the compression results achieved by compressing x, y, p, and t separately using BZip2. The time stamp was compressed using the 4 methods. The best results were

achieved when the time was compressed after taking the consecutive differences without separating the time stamp in 4 bytes that is, using Format 2 as shown in Figure 12. While compressing the polarities were not separated.

In the next part of the experiment, all the components were compressed using the same compression technique separately, but polarities were separated, that is, the polarity bit p was not separately compressed.

In Table 10 - Table 12, the various compression techniques for timestamps as discussed earlier after separating the polarities are represented using the following nomenclature.

1. For method 1, <CompressionTechnique_P> (LZMA_P, PPMD_P, BZIP2_P) is used.
2. For method 2, where the consecutive time differences are encoded, <CompressionTechnique_P_Diff_T> (LZMA_P_DIFF_T, PPMD_P_DIFF_T, BZIP2_P_DIFF_T) is used.
3. For method 3, where the time differences are divided into 4 bytes and then encoded, <CompressionTechnique_P_T_4_bytes> (LZMA_P_T_4_bytes, PPMD_P_T_4_bytes, BZIP2_P_T_4_bytes) is used.
4. For method 4, where the consecutive time differences are separated in 4 bytes and then encoded, <CompressionTechnique_P_Diff_T_4_bytes> (LZMA_P_DIFF_T_4_bytes, PPMD_P_DIFF_T_4_bytes, BZIP2_P_DIFF_T_4_bytes) is used.

Table 10 shows the compression results achieved by compressing x , y , p , and t separately using LZMA. The time stamp was compressed using the 4 methods for polarities 0 and 1 separately. The best results were achieved when the time was compressed after taking the consecutive differences, either with or without separating the time stamp in 4 bytes that is using Format 2 or Format 4 according to Figure 12.

Sequence	LZMA_P	LZMA_P_Diff_T	LZMA_P_T_4_bytes	LZMA_P_Diff_T_4_bytes
Boxes_Rotation	4.9353	5.1382	4.9472	5.138
Dynamic_Rotation	3.4882	3.6328	3.4777	3.643
Outdoors_Running_1	3.6035	3.7681	3.6091	3.7681
Outdoors_Running_2	3.5522	3.7001	3.5452	3.7142
Outdoors_Running_3	3.4087	3.5606	3.3848	3.5662
Outdoors_Walking	3.283	3.4448	3.2466	3.4547
Poster_Rotation	4.7752	4.9863	4.798	4.986
Shapes_Rotation	3.2024	3.3731	3.1791	3.3901
Slider_Close	3.9879	4.1657	3.9153	4.1676
Urban	3.2931	3.4336	3.2505	3.4388

Table 10. Compression results using LZMA on all components separately and separated polarities. The components x and y are encoded in their own sequence, while for encoding the element t , the strategy mentioned in the table's header is utilized after separating polarity 0 and polarity 1 in separated sequences.

Sequence	PPMD_P	PPMD_P_Diff_T	PPMD_P_T_4_bytes	PPMD_P_Diff_T_4_bytes
Boxes_Rotation	3.0926	5.1451	5.118	5.145
Dynamic_Rotation	2.0315	3.4994	3.4576	3.4925
Outdoors_Running_1	2.0941	3.6308	3.5954	3.6308
Outdoors_Running_2	2.0478	3.5736	3.532	3.566
Outdoors_Running_3	1.942	3.4255	3.3767	3.4125
Outdoors_Walking	1.839	3.2903	3.235	3.2808
Poster_Rotation	2.9769	4.96	4.9299	4.96
Shapes_Rotation	1.7624	3.1444	3.0926	3.1384
Slider_Close	2.4305	4.1656	4.1085	4.1637
Urban	1.9315	3.3027	3.2559	3.2929

Table 11. *Compression results using PPMD on all components separately and separated polarities. The components x and y are encoded in their own sequence, while for encoding the element t, the strategy mentioned in the table's header is utilized after separating polarity 0 and polarity 1 in separated sequences.*

Table 11 shows the compression results achieved by compressing x, y, p, and t separately using PPMD. The time stamp was compressed using the 4 methods for polarities 0 and 1 separately. The best results were achieved when the time was compressed after taking the consecutive differences and then separating the time stamp in 4 bytes that is, using Format 4 as shown in Figure 12.

Table 12 shows the compression results achieved by compressing x, y, p, and t separately using BZip2. The time stamp was compressed using the 4 methods for polarities 0 and 1 separately. The best results were achieved when the time was compressed after taking the consecutive differences, either with or without separating the time stamp in 4 bytes that is using the Format 2 or Format 4 according to Figure 12.

Sequence	BZIP2_P	BZIP2_P_Diff_T	BZIP2_P_T_4_bytes	BZIP2_P_Diff_T_4_bytes
Boxes_Rotation	3.6268	4.9422	4.8752	4.942
Dynamic_Rotation	2.1274	3.4558	3.435	3.4656
Outdoors_Running_1	2.2569	3.5964	3.5652	3.5964
Outdoors_Running_2	2.1714	3.5274	3.5068	3.5382
Outdoors_Running_3	2.0037	3.3874	3.3607	3.397
Outdoors_Walking	1.8457	3.255	3.209	3.2589
Poster_Rotation	3.4647	4.7536	4.6908	4.7534
Shapes_Rotation	1.7781	3.1283	3.0856	3.1323
Slider_Close	2.4267	4.0569	3.9798	4.0595
Urban	1.9495	3.2789	3.2394	3.2869

Table 12. *Compression results using BZip2 on all components separately and separated polarities. The components x and y are encoded in their own sequence, while for encoding the element t, the strategy mentioned in the table's header is utilized after separating polarity 0 and polarity 1 in separated sequences.*

From Table 7 - Table 12, we understand the best way to compress the time stamp is to take the consecutive differences and then encode them.

In the next part of the experiment, the number of bits required after compression for each component was calculated. The compressions were done using LZMA, PPMD, and BZip. As seen from Table 7 - Table 12, the best results were achieved when the timestamp was encoded either after taking the consecutive differences or separating the consecutive differences in 4 bytes. So, for the next part of the experiment, only these two methods of encoding t are considered. The comparisons were done without separating the polarities first.

The following nomenclature is used in Table 13 - Table 15 to represent which component is compressed using which compression technique.

1. When a component is compressed using LZMA, $\langle \text{Component}_L \rangle (X_L, Y_L, P_L)$ is used.
2. When a component is compressed using PPMD, $\langle \text{Component}_P \rangle (X_P, Y_P, P_P)$ is used.
3. When a component is compressed using BZIP2, $\langle \text{Component}_B \rangle (X_B, Y_B, P_B)$ is used.

Sequence	X_L	X_P	X_B
Boxes_Rotation	7.5758	7.6038	7.7694
Dynamic_Rotation	7.7883	7.9914	7.947
Outdoors_Running_1	7.811	7.9782	7.943
Outdoors_Running_2	7.8164	7.9926	7.9449
Outdoors_Running_3	7.7625	7.9648	7.9411
Outdoors_Walking	7.7613	7.9587	7.9346
Poster_Rotation	7.5785	7.6395	7.7944
Shapes_Rotation	7.4401	7.8856	7.8976
Slider_Close	7.7128	7.9046	7.9222
Urban	7.7742	7.9751	7.9413

Table 13. *Bits per event required to compress x .*

Table 13 shows the bits per event required to compress the x component. The x component was compressed without separating the polarities (X). The results show that the LZMA compression technique requires the least number of bits to compress the x component which are marked in bold in the table.

Sequence	Y _L	Y _P	Y _B
Boxes_Rotation	2.3655	2.1812	2.5775
Dynamic_Rotation	7.0227	7.2085	7.4093
Outdoors_Running_1	6.7652	6.9016	7.0841
Outdoors_Running_2	6.8719	7.0222	7.1949
Outdoors_Running_3	7.1784	7.3574	7.4375
Outdoors_Walking	6.8264	7.0417	7.2007
Poster_Rotation	2.755	2.6111	3.0371
Shapes_Rotation	7.1391	7.5034	7.5167
Slider_Close	4.0818	3.5622	3.884
Urban	7.2898	7.4568	7.5034

Table 14. *Bits per event required to compress y.*

Table 14 shows the bits per event required to compress the y component. The y component was compressed without separating the polarities (Y). The results show that the LZMA compression technique requires the least number of bits to compress the y component in most of the cases. For Boxes_Rotation, Poster_Rotation, and Slider_Close, PPMD required the least number of bits to compress the y component. The best results are marked in bold.

Table 15 shows the bits per event required to compress the p component. The p component was compressed without separating the polarities (P). The results show that the PPMD compression technique requires the least number of bits to compress the p component which are marked in bold in the table. The result was expected as the polarity bit stream consists of only 2 values 0 and 1, hence the data is highly predictable.

Sequence	P _L	P _P	P _B
Boxes_Rotation	1.0764	0.9992	1.2302
Dynamic_Rotation	1.0773	0.9874	1.221
Outdoors_Running_1	1.0763	0.989	1.2244
Outdoors_Running_2	1.0744	0.9846	1.2207
Outdoors_Running_3	1.0724	0.9911	1.2254
Outdoors_Walking	1.0437	0.9481	1.1776
Poster_Rotation	1.0769	0.9964	1.2288
Shapes_Rotation	1.0696	1.0016	1.2314
Slider_Close	0.9485	0.8351	1.0555
Urban	1.0599	0.9589	1.1998

Table 15. *Bits per event required to compress p.*

The following nomenclature is used in Table 16 to represent which method of compressing the timestamp is used in addition to the compression technique.

1. When a timestamp is compressed after taking the consecutive difference, $\langle T_D_{\text{CompressionTechnique}} \rangle (T_D_L, T_D_P, T_D_B)$ is used.
2. When a timestamp is compressed after separating the consecutive differences in 4 bytes, $\langle T_D4_{\text{CompressionTechnique}} \rangle (T_D4_L, T_D4_P, T_D4_B)$ is used.

Table 16 shows the bits per event required to compress the timestamp t . The timestamp was compressed after taking the consecutive differences (T_D) and separating the time differences in 4 bytes (T_D4) without separating the polarities. The results show that the PPMD compression technique requires the least number of bits to compress the timestamp. The least number of bits required to compress was achieved when the consecutive differences were encoded for most of the sequences. For Dynamic_Rotation the least number of bits to encode the time stamp was achieved after separating the consecutive time differences in 4 bytes. The best results are marked in bold in the table.

Sequence	T_D_L	T_D_P	T_D_B	T_D4_L	T_D4_P	T_D4_B
Boxes_Rotation	0.8221	0.6737	0.8457	0.8257	0.6825	0.8457
Dynamic_Rotation	1.9708	1.7303	2.1162	1.8448	1.7016	2.0828
Outdoors_Running_1	1.4326	1.2562	1.5858	1.4362	1.2651	1.5858
Outdoors_Running_2	1.6562	1.4945	1.845	1.6598	1.5034	1.8451
Outdoors_Running_3	2.2826	2.0394	2.5112	2.1971	2.0683	2.4462
Outdoors_Walking	3.2842	3.079	3.5496	3.2608	3.1636	3.4932
Poster_Rotation	0.8329	0.6779	0.8627	0.8365	0.6867	0.8628
Shapes_Rotation	3.6846	3.5129	4.0037	3.6663	3.6224	3.9464
Slider_Close	2.1772	2.0336	2.4248	2.1823	2.0429	2.4259
Urban	2.7516	2.6207	3.0449	2.7555	2.6297	3.0452

Table 16. *Bits per event required to compress t .*

The number of bits required to compress each component was again calculated after separating the polarities in the next part. X^0 and X^1 were compressed separately and then combined while calculating the total number of bits required. The same process was repeated for y and t . The data for polarity is inherently encoded, hence the polarity bit p was not separately encoded for this part of the experiment.

The following nomenclature is used in Table 17 and Table 18 to represent which component is compressed using which compression technique after the polarities were separated.

1. When a component is compressed using LZMA, $\langle \text{Component_P}_L \rangle (X_{P_L}, Y_{P_L})$ is used.
2. When a component is compressed using PPMD, $\langle \text{Component_P}_P \rangle (X_{P_P}, Y_{P_P})$ is used.
3. When a component is compressed using BZIP2, $\langle \text{Component_P}_B \rangle (X_{P_B}, Y_{P_B})$ is used.

Sequence	X _{PL}	X _{PP}	X _{PB}
Boxes_Rotation	7.8097	8.0793	7.9543
Dynamic_Rotation	7.749	8.1946	7.9839
Outdoors_Running_1	7.7791	8.1712	7.9805
Outdoors_Running_2	7.7801	8.1942	7.9841
Outdoors_Running_3	7.7228	8.1547	7.977
Outdoors_Walking	7.7074	8.1576	7.9742
Poster_Rotation	7.7616	8.0528	7.945
Shapes_Rotation	7.3426	8.0774	7.9501
Slider_Close	7.6034	8.1057	7.9433
Urban	7.7137	8.1837	7.9783

Table 17. *Bits per event required to compress x with polarities separated.*

Table 17 shows the bits per event required to compress the x component. The x component was compressed after separating the polarities (X_P). The results show that the LZMA compression technique requires the least number of bits to compress the x which are marked in bold.

Sequence	Y _{PL}	Y _{PP}	Y _{PB}
Boxes_Rotation	3.3404	3.1463	3.5471
Dynamic_Rotation	7.1114	7.5115	7.4926
Outdoors_Running_1	6.9311	7.2744	7.2747
Outdoors_Running_2	6.9336	7.3097	7.3232
Outdoors_Running_3	7.1313	7.5446	7.4858
Outdoors_Walking	6.7088	7.2144	7.2328
Poster_Rotation	3.7168	3.59	4.0006
Shapes_Rotation	6.9926	7.6931	7.5585
Slider_Close	4.7406	4.2964	4.4484
Urban	7.286	7.6834	7.5533

Table 18. *Bits per event required to compress y with polarities separated.*

Table 18 shows the bits per event required to compress the y component. The y component was compressed after separating the polarities (Y_P). The results show that the LZMA compression technique requires the least number of bits to compress the y component in most of the cases. For Boxes_Rotation, Poster_Rotation, and Slider_Close, PPMD required the least number of bits to compress the y component. The best results are marked in bold.

The following nomenclature is used in Table 19 to represent which method of compressing the timestamp is used in addition to the compression technique.

1. When a timestamp is compressed after taking the consecutive difference, $\langle T_P_D_{\text{CompressionTechnique}} \rangle (T_P_D_L, T_P_D_P, T_P_D_B)$ is used.
2. When a timestamp is compressed after separating the consecutive differences in 4 bytes, $\langle T_P_D4_{\text{CompressionTechnique}} \rangle (T_P_D4_L, T_P_D4_P, T_P_D4_B)$ is used.

Table 19 shows the bits per event required to compress the timestamp t . The timestamp was compressed after taking the consecutive differences and separating the time differences in 4 bytes after separating the polarities. The results show that the PPMD compression technique for compressing the consecutive differences requires the least number of bits to compress the timestamp for most of the sequences. For Shapes_Rotation the least number of bits to encode the time stamp was achieved after separating the consecutive time differences in 4 bytes using the LZMA technique. The best results are marked in bold in the table.

Sequence	T_P_D _L	T_P_D _P	T_P_D _B	T_P_D4 _L	T_P_D4 _P	T_P_D4 _B
Boxes_Rotation	1.3233	1.2186	1.4505	1.3244	1.2189	1.4511
Dynamic_Rotation	2.7523	2.5913	3.0429	2.7066	2.6255	2.9896
Outdoors_Running_1	2.2779	2.1812	2.5384	2.2787	2.1814	2.5388
Outdoors_Running_2	2.5395	2.4193	2.805	2.5196	2.4404	2.7798
Outdoors_Running_3	3.1225	2.9918	3.4317	3.0948	3.0604	3.379
Outdoors_Walking	4.1655	4.0844	4.4449	4.1046	4.1328	4.4235
Poster_Rotation	1.3713	1.2662	1.5089	1.3725	1.2665	1.5095
Shapes_Rotation	4.6391	4.61	4.945	4.532	4.642	4.9204
Slider_Close	3.0598	3.0072	3.3744	3.0616	3.0177	3.3679
Urban	3.6124	3.5459	3.9668	3.6006	3.5805	3.9347

Table 19. *Bits per event required to compress t with polarities separated.*

In the next part of this experiment, we will compress the individual components with the best compression techniques suited for that component as recorded in Tables 13 - 16 when the polarities are not separated and Table 17 - 19 when the polarities are separated. All the components were then combined using the LZMA compression technique.

Table 20 shows the compression ratio achieved by selecting the best compression method for individual components without separating the polarities as understood from Tables 13 - 16. A Proposed Default Method 1 was then suggested where the spatial address x and y are compressed using LZMA, the polarity bit p is compressed using PPMD and the timestamp t was compressed after taking consecutive time differences using PPMD. These individual components were then combined in a single file using the LZMA compression technique. The Default Method 1 gives the result almost as good as the achieved results.

Sequence	X	Y	P	T	Compression Ratio	Proposed Default Method 1 X_L, Y_L, P_P, T_{D_P}
Boxes_Rotation	X_L	Y_P	P_P	T_{D_P}	5.5989	5.5101
Dynamic_Rotation	X_L	Y_L	P_P	T_{D4_P}	3.6587	3.6509
Outdoors_Running_1	X_L	Y_L	P_P	T_{D_P}	3.8044	3.8044
Outdoors_Running_2	X_L	Y_L	P_P	T_{D_P}	3.7277	3.7277
Outdoors_Running_3	X_L	Y_L	P_P	T_{D_P}	3.561	3.561
Outdoors_Walking	X_L	Y_L	P_P	T_{D_P}	3.4378	3.4378
Poster_Rotation	X_L	Y_P	P_P	T_{D_P}	5.3941	5.3295
Shapes_Rotation	X_L	Y_L	P_P	T_{D_P}	3.3516	3.3516
Slider_Close	X_L	Y_P	P_P	T_{D_P}	4.5239	4.3637
Urban	X_L	Y_L	P_P	T_{D_P}	3.4325	3.4325

Table 20. *Compression Ratios obtained by compressing each component separately and using different compression techniques without separating polarities.*

Table 21 shows the compression ratio achieved by selecting the best compression method for individual components after compressing the polarities as understood from tables 16-18. A proposed Default Method 2 was then suggested where the spatial address x and y are compressed using LZMA, the polarity bit p is not explicitly compressed as the polarity information is linked with the other components. The timestamp t was compressed after taking consecutive time differences using PPMD. These individual components were then combined in a single file using the LZMA compression technique. The Default Method 2 gives the result almost as good as the achieved results.

Sequence	X	Y	T	Compression Ratio	Proposed Default Method 2 $X_{P_L}, Y_{P_L}, T_{P_{D_P}}$
Boxes_Rotation	X_{P_L}	Y_{P_P}	$T_{P_{D_P}}$	5.3004	5.1859
Dynamic_Rotation	X_{P_L}	Y_{P_L}	$T_{P_{D_P}}$	3.6797	3.6797
Outdoors_Running_1	X_{P_L}	Y_{P_L}	$T_{P_{D_P}}$	3.8003	3.8003
Outdoors_Running_2	X_{P_L}	Y_{P_L}	$T_{P_{D_P}}$	3.7476	3.7476
Outdoors_Running_3	X_{P_L}	Y_{P_L}	$T_{P_{D_P}}$	3.6006	3.6006
Outdoors_Walking	X_{P_L}	Y_{P_L}	$T_{P_{D_P}}$	3.4769	3.4769
Poster_Rotation	X_{P_L}	Y_{P_P}	$T_{P_{D_P}}$	5.1172	5.0331
Shapes_Rotation	X_{P_L}	Y_{P_L}	$T_{P_{D4_L}}$	3.3919	3.3976
Slider_Close	X_{P_L}	Y_{P_P}	$T_{P_{D_P}}$	4.3581	4.1936
Urban	X_{P_L}	Y_{P_L}	$T_{P_{D_P}}$	3.4678	3.4678

Table 21. *Compression Ratios obtained by compressing each component separately and using different compression techniques after separating polarities.*

Table 22 compares the results of the Proposed Default Method 1, Proposed Default Method 2 and previously achieved results by Khan et. al., in [4] and Bi et al. in [2]. There was an average 13.52% increase in compression ratio when compared to results achieved by Khan et. al. in [2]. When compared to the results achieved by Bi et. al., in [2], we understand that 3D encoders give better results than the 1D encoders. The best results in the 1D encoders are marked in bold and the best results from 1D and 3D encoders are highlighted in green in the table.

Figure 13 summarizes the comparison of 1D encoder results in visual format.

Sequence	1D Encoders				3D Encoder
	LZMA (Access 2020) [4]	Proposed Default Method 1	Proposed Default Method 2	Percentage Improvement	Spike Coding (DCC 2018) [2]
Boxes_Rotation	4.92	5.5101	5.1859	11.9939	5.35
Dynamic_Rotation	3.34	3.6509	3.6797	10.1707	4.99
Outdoors_Running_1	3.25	3.8044	3.8003	17.0585	3.97
Outdoors_Running_2	3.41	3.7277	3.7476	9.90029	4.46
Outdoors_Running_3	3.49	3.561	3.6006	3.16905	4.67
Outdoors_Walking	3.11	3.4378	3.4769	11.7974	3.84
Poster_Rotation	4.77	5.3295	5.0331	11.7296	5.44
Shapes_Rotation	3.04	3.3516	3.3976	11.7632	4.39
Slider_Close	3.19	4.3637	4.1936	36.7931	3.62
Urban	3.13	3.4325	3.4678	10.7923	3.66
	Average percentage increase:			13.5168	

Table 22. Comparison of Compression Ratio from previous work and results achieved in this experiment.

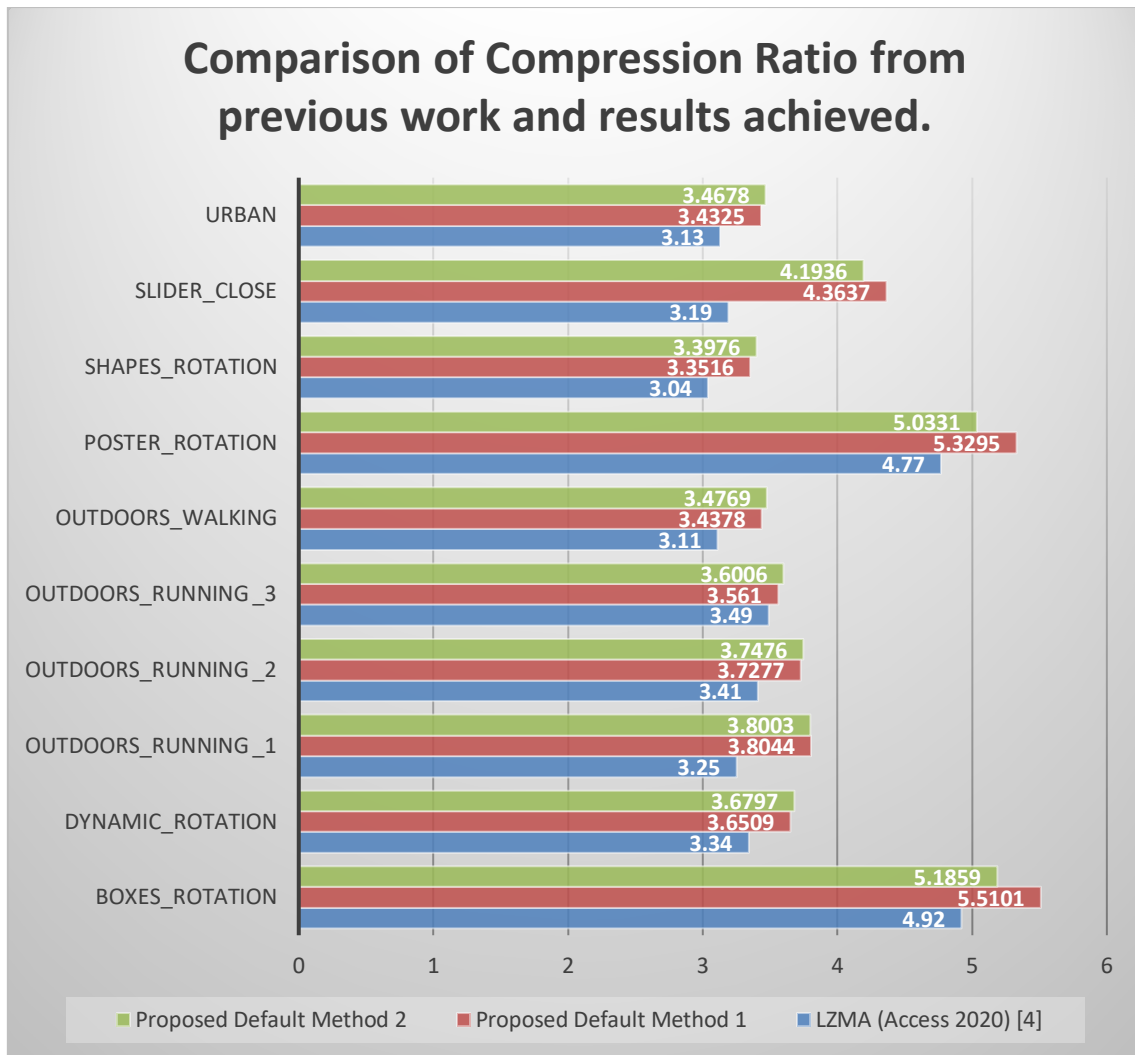


Figure 13. Comparison of Compression Ratio from previous work and results achieved

3.3 Lossless Compression of DAVIS Camera Event Data using PPMD Encoded Time Differences and GPCC Encoder

In this section, we propose several new algorithms that combine the 3-Dimensional encoder G-PCC and the 1-Dimensional encoder that we found to be the most efficient for encoding differential time stamps. We also compare the achieved results with our re-implementation of the method presented in [7]. We noticed that our re-implementation provides different results than those reported in [7]. One main reason is that the new version of G-PCC is different from the old version since the set of options available in the two programs are slightly different. Given this impossibility of reproducing the results of [7], we compare the new algorithm that we propose with our re-implementation to find the best algorithmic versions. For completeness in the end, we also show the results compared to those from [7], but noting again there is a difference in the baseline G-PCC codecs used in [7] (v2 [21]) and the most recent version (v12

[22]). In addition to the disabled options, the GPCC encoder can handle only single-point floating values, while the timestamp values extracted from the AER data are so large that they need a double floating-point format. Thus, the timestamp cannot be given as the third coordinate in the point cloud without some processing. To overcome this issue, we encoded differential time stamps separately and the indices of the events were used as the third axis for Point cloud compression.

Figure 14 shows the flow of data.

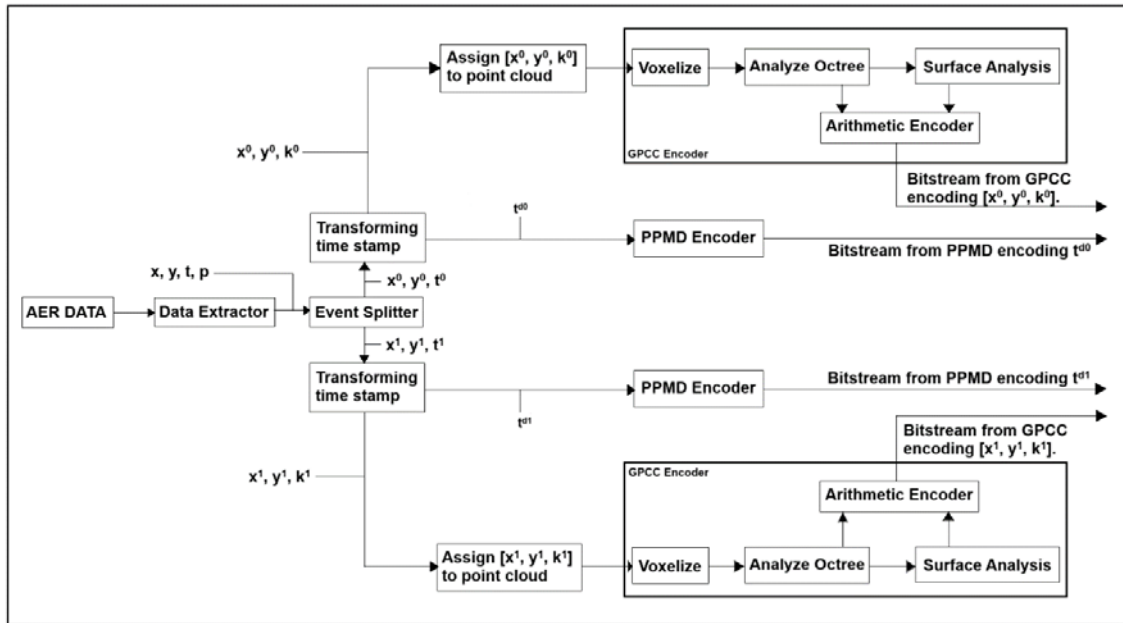


Figure 14. Lossless compression of DAVIS camera event data using PPMD encoded time differences and GPCC encoder.

Algorithm 2 tells the steps used to perform GPCC encoding in this thesis. The AER data is initially extracted to get the x , y , t , and p components.

The events with polarities 0 and 1 are then separated. After the data is split, the time stamps are represented as consecutive time differences and encoded by using the PPMD encoder. The indices k of the events are then given to the GPCC encoder as the third axis for the point cloud. GPCC encoding is then performed on it and the encoded bitstream is given as the output. This process is repeated for the data with both polarities. The output of this method is GPCC encoded bitstream for polarities 0 and 1 and the t^d PPMD encoded bitstream for polarities 0 and 1.

Algorithm 2: Point Cloud Compression algorithm to perform GPCC encoding.

Input: Spike event data in AER.

Total number of events N_{events} .

Number of input bits: $N_{input} = N_{events} \times 64$.

Step 1: Extract data \mathbf{x} , \mathbf{y} , \mathbf{t} , and \mathbf{p} from AER data

Step 2: Split data into events of polarities 0 and 1 separately $(\mathbf{x}^0, \mathbf{y}^0, \mathbf{t}^0, \mathbf{p}^0)$, $(\mathbf{x}^1, \mathbf{y}^1, \mathbf{t}^1, \mathbf{p}^1)$.

Step 3: Calculate $\mathbf{t}^{d0} = \mathbf{t}_{2:N_{events}}^0 - \mathbf{t}_{1:(N_{events}-1)}^0$ for events with polarity 0.

Step 4: Encode \mathbf{t}^{d0} using the PPMD encoder.

Step 5: Compute \mathbf{k}^0 as the indices of events with polarity zero.

Step 6: Create a point cloud with $[\mathbf{x}^0, \mathbf{y}^0, \mathbf{k}^0]$.

Step 7: Apply GPCC encoding to the point cloud from step 6.

Step 8: Repeat steps 3 to 7 for events with polarity one.

Output: Bitstream from PPMD encoding of \mathbf{t}^{d0} , with size N_{P0} in bits

Output: Bitstream from GPCC encoding $[\mathbf{x}^0, \mathbf{y}^0, \mathbf{k}^0]$, with size N_{G0} in bits

Output: Bitstream from PPMD encoding of \mathbf{t}^{d1} , with size N_{P1} in bits

Output: Bitstream from GPCC encoding $[\mathbf{x}^1, \mathbf{y}^1, \mathbf{k}^1]$, with size N_{G1} in bits

Output: Total size in bits of the concatenated output bitstream, $\gamma = N_{P0} + N_{G0} + N_{P1} + N_{G1}$

Output: *Compression Ratio* = $\frac{N_{input}}{\gamma}$

This algorithm was applied over two configurations for the GPCC codec [22].

The first configuration was similar to the configuration used by Martini et.al., in [7] as shown in Table 2. Since some of the options were disabled, we omitted them and kept the other options the same.

In the second configuration, we modified the option **maxNumQtBtBeforeOt** from 4 to 24. This option limits the number of quad and binary trees in the first partition of the octree. We also changed the option **minQtbtSizeLog2** from 0 to 2. This option tells the minimum size of quad and binary tree partitions. We added the option **qtbtEnabled** which enables non-cubic geometry coding. By adding the option **bitwiseOccupancyCoding**, we enable bitwise coding of the octree.

The two configurations are summarized in Table 23.

Option	Configuration 1	Configuration 2
mergeDuplicatedPoints	0	0
planarEnabled	1	1
planarModelDcmUse	0	-1
positionQuantizationScale	1	1
trisoupNodeSizeLog2	0	0
neighbourAvailBoundaryLog2	8	8
intra_pred_max_node_size_log2	6	6
inferredDirectCodingMode	1	1
maxNumQtBtBeforeOt	4	24
minQtbtSizeLog2	0	3
sliceMaxPoints	107000000	107000000
convertPlyColourspace	1	1
transformType	0	0
numberOfNearestNeighborsInPrediction	3	3
levelOfDetailCount	12	12
intraLodPredictionSkipLayers	0	0
interComponentPredictionEnabled	0	0
positionBaseQp	0	0
enforceLevelLimits	0	0

Table 23. *Configurations used for the GPCC encoder. On the left is the configuration that we adapted from [7] while on the right is the configuration that we found the most efficient with the current version of GPCC codec [22].*

In addition to the above two methods, we also tried various permutations of the x, y, and the indices, to understand the role of the axes in the resulting compression ratios. Table 24 depicts the assignment of the x, y, and z axes to the x, y, and k components used in the implementation of the code.

Permutations	X axis	Y axis	Z axis
1	k	x	y
2	k	y	x
3	y	k	x
4	y	x	k

Table 24. *Permutations used for assigning the components to the axes.*

Sequences	Original number of events	Number of events compressed	Percentage of events compressed	Compression Ratio
Boxes_Rotation	21443227	21443227	100	4.9095
Dynamic_Rotation	21554500	17404211	80.7451	4.1939
Outdoors_Running_1	30509927	26706841	87.5349	4.1641
Outdoors_Running_2	24587989	18744456	76.2342	4.0566
Outdoors_Running_3	14276086	11826956	82.8445	3.9979
Outdoors_Walking	6843927	5393553	78.8079	3.8172
Poster_Rotation	20105490	20105490	100	5.0211
Shapes_Rotation	4912273	3769614	76.7387	4.4376
Slider_Close	1380337	1380337	100	4.1558
Urban	5030400	5030400	100	3.7267

Table 25. Compression ratio achieved using the configuration method 1 adapted from Martini et. al. in [7].

Table 25 shows the compression ratio obtained by having the configuration of the GPCC encoder taken from the configuration published by Martini et. al., in [7]. The GPCC encoder can only handle values as a single floating point. To achieve lossless compression, the sequences were compressed partially. The maximum number of events that could be decoded without losing the time stamp data was compressed. These are tabulated under the column “Number of events compressed”. For sequences Boxes_Rotation, Poster_Rotation, Slider_Close, and Urban the complete sequence was compressed.

Sequences	Results achieved using GPCC codec [22]	Point Cloud Compression (IEEE ACCESS 2022) [7] used GPCC codec [21]
Boxes_Rotation	4.91	5.35
Dynamic_Rotation	4.19	4.99
Outdoors_Running_1	4.16	3.97
Outdoors_Running_2	4.06	4.46
Outdoors_Running_3	4	4.67
Outdoors_Walking	3.82	3.84
Poster_Rotation	5.02	5.44
Shapes_Rotation	4.44	4.39
Slider_Close	4.16	3.62
Urban	3.73	3.66

Table 26. Comparison of results achieved in this thesis with GPCC codec [22] and results achieved by Martini et. al., in [7]

Table 26 compares the results achieved in this thesis and the results achieved by Martini et. al., in [7]. Since some of the configurations are now disabled in the GPCC encoder and partial sequences were compressed, the results were not matched. In some cases, the achieved results exceeded the results achieved by Martini. The best results are marked in bold.

Table 27 tabulates the results from the proposed method, where the consecutive time stamps were compressed using PPMD after separating polarities, and the spatial address x , y and the event index k were encoded using GPCC. Since the GPCC encoder can handle values of the resolution of single float values that is values up to 2^{24} (16 777 216), while compressing the polarity 0 values in `Outdoors_Running_1`, we had to truncate the sequence as it went above the permissible limit. All the results henceforth for `Outdoors_Running_1` are for 29 541 919 events out of the 30 509 927 total events which is 96.83% of total events. This was done to avoid lossy compression of the sequence while calling the GPCC.

The best results are achieved using the indices k as the first axis, component y as the second, and x as the third axis and configuration 1 for GPCC. The consecutive differences are compressed using PPMD. For `Boxes_Rotation`, configuration 2 for GPCC and the point cloud having y as the first axis, x as the second axis and the indices k as the third axis gave the best results.

Sequences	Configuration 1				Configuration 2				Percentage sequence compressed
	1	2	3	4	1	2	3	4	
<code>Boxes_Rotation</code>	4.0545	4.1208	4.1163	4.1028	4.1562	4.1546	4.1581	4.1616	100
<code>Dynamic_Rotation</code>	3.7581	3.7584	3.7564	3.7531	3.3805	3.3803	3.3837	3.3862	100
<code>Outdoors_Running_1</code>	3.7106	3.7134	3.7116	3.7094	3.4593	3.459	3.4606	3.4632	96.83
<code>Outdoors_Running_2</code>	3.6576	3.6598	3.6589	3.6576	3.4203	3.4201	3.4218	3.4252	100
<code>Outdoors_Running_3</code>	3.5944	3.5958	3.595	3.5934	3.3011	3.3013	3.3033	3.3075	100
<code>Outdoors_Walking</code>	3.4638	3.4643	3.4628	3.4612	3.1978	3.1984	3.2011	3.2049	100
<code>Poster_Rotation</code>	4.1474	4.2098	4.204	4.1899	4.0978	4.0962	4.1008	4.1027	100
<code>Shapes_Rotation</code>	3.977	3.977	3.9737	3.9724	3.3165	3.3165	3.3207	3.3238	100
<code>Slider_Close</code>	3.7137	3.7456	3.7429	3.7205	3.6214	3.6197	3.6204	3.6245	100
<code>Urban</code>	3.3644	3.3657	3.3657	3.3633	3.143	3.1431	3.1464	3.1488	100

Table 27. Compression ratios achieved using the 2 configurations and 4 permutations of the axes.

Moving forward, PPMD + GPCC will refer to the results achieved with GPCC configuration 1 and indices k as the first axis, component y as the second, and x as the third axis. All the PPMD + GPCC results henceforth for `Outdoors_Running_1` are achieved for 96.83% of the total events as the GPCC encoder can handle values only for single point floating resolution and the polarity 0 events exceeded the permissible limit. To avoid lossy compression, the sequence was truncated.

Sequences	Original Number of events	Configuration taken from Martini et. al., in [7]		PPMD + GPCC	
		Number of events compressed	Percentage events compressed	Number of events compressed	Percentage events compressed
Boxes_Rotation	21443227	21443227	100	21443227	100
Dynamic_Rotation	21554500	17404211	80.75	21554500	100
Outdoors_Running_1	30509927	26706841	87.53	29541919	96.83
Outdoors_Running_2	24587989	18744456	76.23	24587989	100
Outdoors_Running_3	14276086	11826956	82.84	14276086	100
Outdoors_Walking	6843927	5393553	78.81	6843927	100
Poster_Rotation	20105490	20105490	100	20105490	100
Shapes_Rotation	4912273	3769614	76.74	4912273	100
Slider_Close	1380337	1380337	100	1380337	100
Urban	5030400	5030400	100	5030400	100

Table 28. Comparison of number of events compressed using configuration taken from Martini et. al. In [7], and PPMD + GPCC

Table 28 compares the number of events compressed using the compression techniques and configurations similar to Martini et. al., in [7] and PPMD + GPCC compression. Using the PPMD + GPCC compression method, all the data sequences except Outdoor_running_1 were compressed completely. In Outdoor_running_1, the number of events compressed has increased in PPMD+GPCC from 87.53% to 96.83%.

Sequences	PPMD + GPCC	Compression by GPCC of truncated event sequences	Point Cloud Compression (IEEE ACCESS 2022) [7]
Boxes_Rotation	4.1208	4.9095	5.35
Dynamic_Rotation	3.7584	4.1939	4.99
Outdoors_Running_1	3.7134	4.1641	3.97
Outdoors_Running_2	3.6598	4.0566	4.46
Outdoors_Running_3	3.5958	3.9979	4.67
Outdoors_Walking	3.4643	3.8172	3.84
Poster_Rotation	4.2098	5.0211	5.44
Shapes_Rotation	3.977	4.4376	4.39
Slider_Close	3.7456	4.1558	3.62
Urban	3.3657	3.7267	3.66

Table 29. Comparison of results achieved using the proposed method, compressing the partial results with configuration same as Martini et. al., and results achieved by Martini et. al., in [7]

Table 29 compares the results achieved using PPMD + GPCC compression, compression of partial events using the GPCC configuration from Martini et.al., in [7] and results achieved by Martini et. al., in [7]. The best results are marked in bold. The results achieved by us using the PPMD + GPCC exceeded in the sequence Slider_Close than the results achieved in [7].

Considering results after compressing event data using 1-D encoders and 3D encoder proposed in this thesis, Table 30 is formulated. As seen from this table, results obtained from 1D

encoders are better for almost every sequence than the results obtained using the PPMD+GPCC encoder. We note that all the results obtained in this table are verified for perfect lossless reconstruction and are obtained on the same data. Table 30 also compares the compression + decompression time required by the encoders. The 3D encoders are on average 60 times slower than the 1D encoders, making 1D encoders more suited for real-time applications.

Sequence	Compression Ratios			Sequence duration (s)	Compression + Decompression Time		
	Proposed Method 1	Proposed Method 2	PPMD + GPCC		Proposed Method 1	Proposed Method 2	PPMD + GPCC
Boxes_Rotation	5.5101	5.1859	4.1208	5	11.5	13.2564	801.9913
Dynamic_Rotation	3.6509	3.6797	3.7584	20	13.801	10.9433	790.1254
Outdoors_Running_1	3.8044	3.8003	3.6832	20	16.4405	16.4158	1149.8224
Outdoors_Running_2	3.7277	3.7476	3.6598	20	13.5265	13.2171	906.581
Outdoors_Running_3	3.561	3.6006	3.5958	20	7.7926	7.8524	508.6693
Outdoors_Walking	3.4378	3.4769	3.4643	20	4.4948	4.8842	233.2703
Poster_Rotation	5.3295	5.0331	4.2098	5	11.6461	13.3347	748.8237
Shapes_Rotation	3.3516	3.3976	3.977	20	3.6116	5.6343	161.2724
Slider_Close	4.3637	4.1936	3.7456	3	2.0504	2.3815	46.0807
Urban	3.4325	3.4678	3.3657	10	3.964	4.1704	175.8259

Table 30. Comparison of Compression ratio and Compression + Decompression Times of 1D and 3D encoders

Considering Table 22 and Table 29, and the results achieved by Martini et.al., in [7], Table 31 is formulated. However, we stress again that the results of [7] are obtained with a version of GPCC that is not available anymore, and hence we could not reproduce their results.

Sequence	Previously Achieved Results			Results achieved in thesis		
	LZMA (Access 2020) [4]	Spike Coding (DCC 2018) [2]	Point Cloud Compression (Access 2022) [7]	Proposed Method 1	Proposed Method 2	PPMD + GPCC
Boxes_Rotation	4.92	4.95	5.35	5.5101	5.1859	4.1208
Dynamic_Rotation	3.34	3.85	4.99	3.6509	3.6797	3.7584
Outdoors_Running_1	3.25	3.68	3.97	3.8044	3.8003	3.7134
Outdoors_Running_2	3.41	3.92	4.46	3.7277	3.7476	3.6598
Outdoors_Running_3	3.49	3.97	4.67	3.561	3.6006	3.5958
Outdoors_Walking	3.11	3.54	3.84	3.4378	3.4769	3.4643
Poster_Rotation	4.77	4.88	5.44	5.3295	5.0331	4.2098
Shapes_Rotation	3.04	3.78	4.39	3.3516	3.3976	3.977
Slider_Close	3.19	3.84	3.62	4.3637	4.1936	3.7456
Urban	3.13	3.45	3.66	3.4325	3.4678	3.3657

Table 31. Comparison of compression ratio of previously achieved results and results achieved by proposed methods.

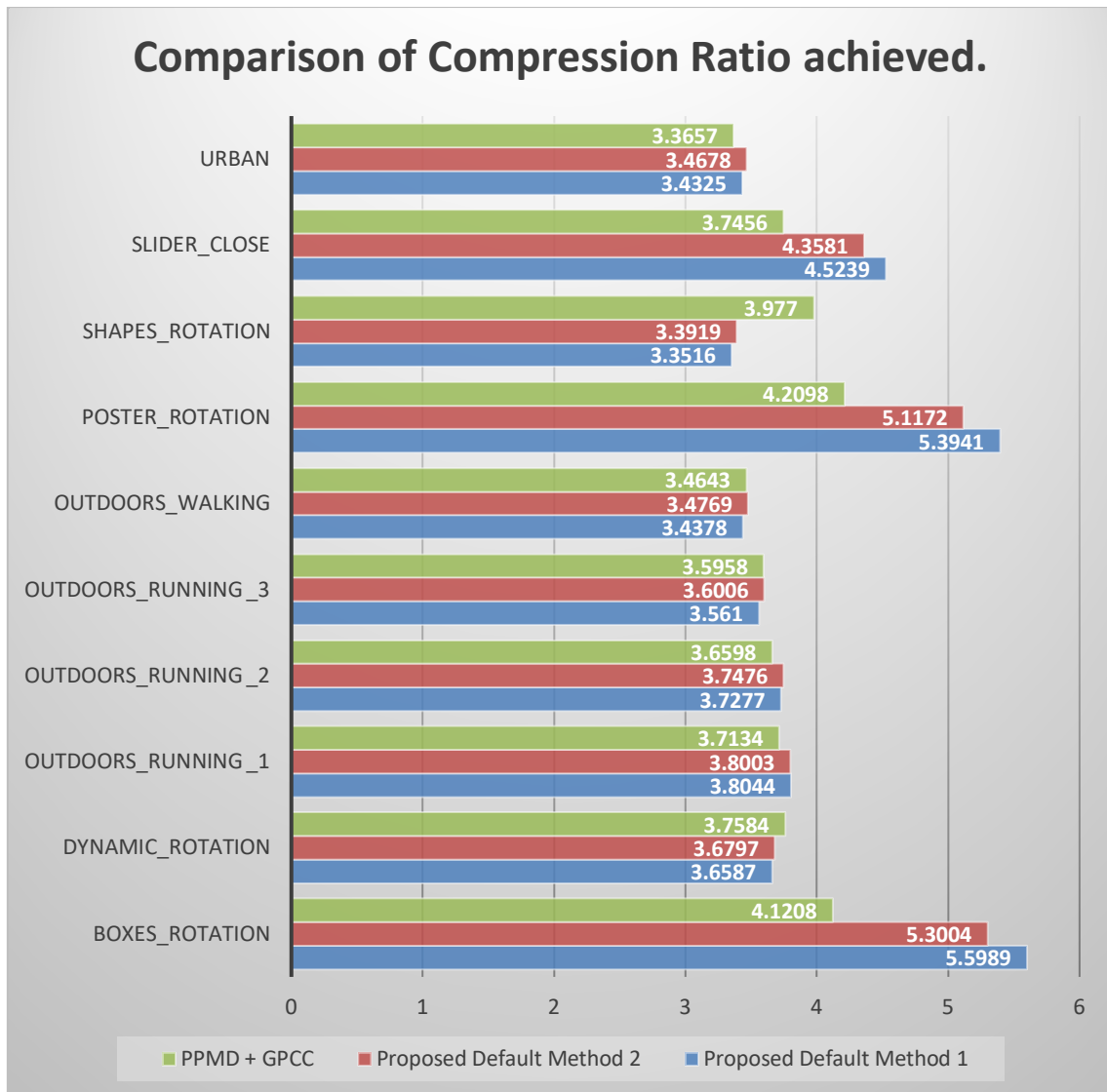


Figure 15. Comparison of Compression Ratio achieved using 1D encoders and 3D encoders in visual format.

Figure 15 summarizes the comparison of results achieved using the proposed 1D and 3D encoders in visual format.

Table 32 compares the compression speed and decompression speed of 1D encoders. The values are in MB/s. The average compression speed required by proposed method 1 is the fastest, while the average decompression speed achieved by Khan et. al., in [4] using LZMA is the fastest.

Figure 16 and Figure 17 summarize the comparison of compression and decompression speeds of the State-of-the-art 1D encoder and specialized event data encoder with the results achieved by using the proposed 1D encoders in this thesis visually.

Sequences	Compression Speed (MB/s)				Decompression Speed (MB/s)			
	Spike Coding (DCC 2018) [2]	LZMA (Access 2020) [4]	Proposed Method 1	Proposed Method 2	Spike Coding (DCC 2018) [2]	LZMA (Access 2020) [4]	Proposed Method 1	Proposed Method 2
Boxes_Rotation	9.45	9.87	24.15	21.63	74.25	77.3	38.61	32.21
Dynamic_Rotation	8.21	8.35	17.98	21.88	52.25	54.7	40.95	56.31
Outdoors_Running_1	8.05	8.21	19.88	21.28	48.65	53.65	58.63	49.37
Outdoors_Running_2	8.25	8.52	19.98	20.68	54.28	56.19	53.46	53.1
Outdoors_Running_3	8.38	8.37	21.13	20.41	55.18	56.54	47.84	50.62
Outdoors_Walking	7.75	8.36	17.42	16.07	47.97	51.37	40.53	37.09
Poster_Rotation	9.38	9.72	23.16	20.18	73.11	75.12	34.22	29.98
Shapes_Rotation	7.92	8.23	16.07	9.014	47.33	49.83	33.71	30.83
Slider_Close	9.25	9.9	8.6	7.17	49.47	50.69	14.4	13.13
Urban	7.87	8.5	15.53	13.98	47.89	51.08	29.33	31.14
Average:	8.451	8.803	18.39	17.23	55.038	57.647	39.17	38.38

Table 32. Comparison of compression and decompression speeds of previously achieved results and results achieved by proposed methods.

The Compression Speed (CS) and the Decompression Speed (DS) are calculated using the formulas given below where T_C is the time required for compression and T_D is the time required for decompression.

$$CS = \frac{N_{events} \times 8}{T_c} \text{ bytes/s}$$

$$DS = \frac{N_{events} \times 8}{T_D} \text{ bytes/s}$$

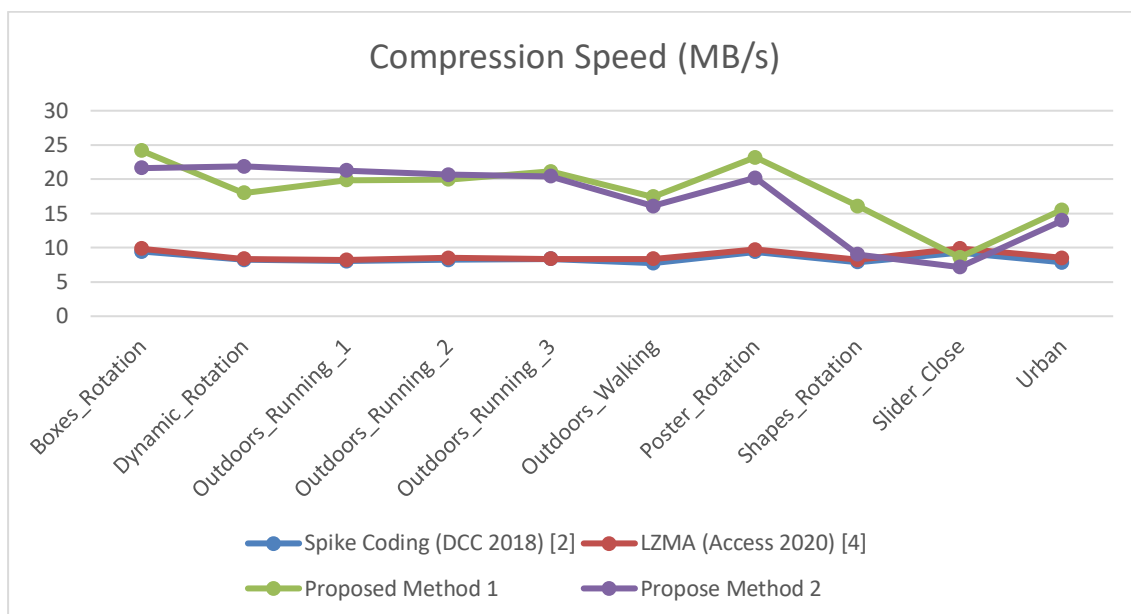


Figure 16. Comparison of compression Speeds from previously achieved results and results achieved by proposed methods.

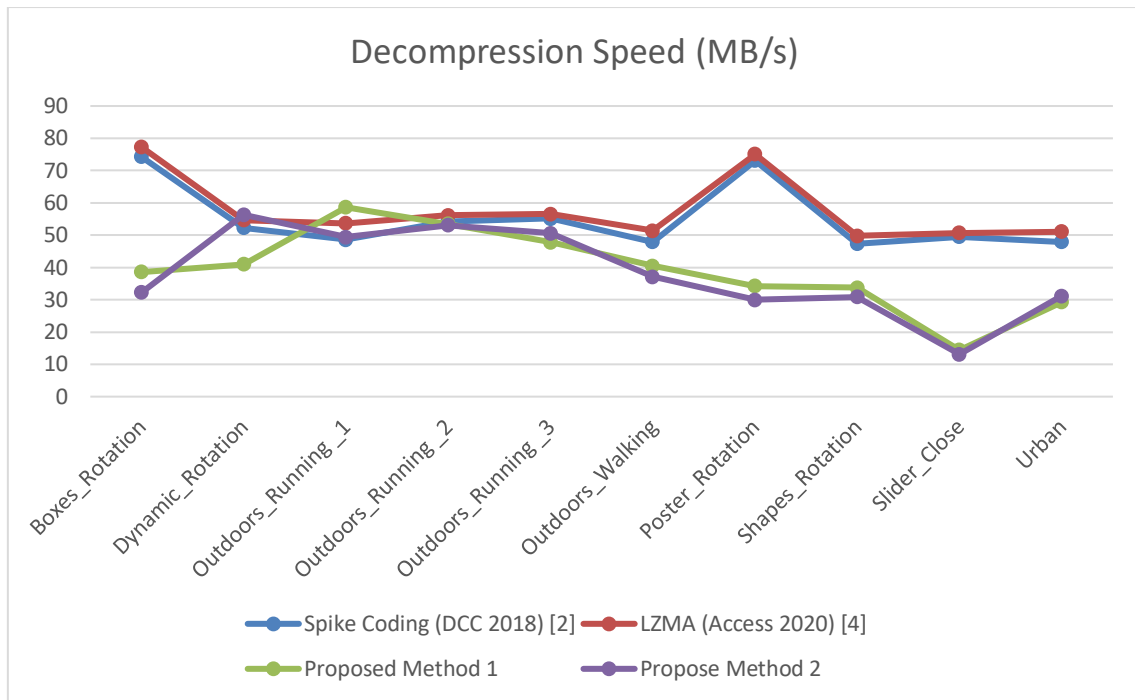


Figure 17. Comparison of Decompression Speeds from previously achieved results and results achieved by proposed methods.

4. CONCLUSIONS

In this thesis, we explore various compression techniques that could potentially compress the event data in a lossless manner. Event cameras have been introduced quite recently in the world of electronics, and they can capture data at a higher temporal resolution, higher dynamic resolution and with low power consumption. The high temporal resolution results in a higher amount of data, which lowers the efficiency with which the data can be stored and transmitted. This thesis focussed on overcoming this issue by the use of lossless compression techniques.

In the first part of the thesis, we investigated how the existing compression techniques were used by doing a literature survey. With this survey, we understood what techniques were relatively effective and the limitations these techniques had. We also studied the advantages and the limitations of the state-of-the-art techniques developed by the researchers.

In the second part of the thesis, we investigated the use of the 1D encoders available at our disposal and found the combinations that provided the best results. We compressed the individual event data components with a compression technique that required the least number of bits per event without having to separate the polarities. Using this method, we were able to achieve results almost as good as the state-of-the-art encoding techniques.

In the next part of the thesis, we compared the results of the best combinations of 1D algorithms when encoding separately the files of a given event polarity, which made the encoding of the “polarity” element of the data unnecessary. The results achieved were as good as the results achieved without separating the polarities, the compression and decompression speed were also almost the same.

In the last part of the thesis, we shifted our focus to the 3D encoder GPCC. After understanding the limitations of the existing implementation of the GPCC encoder, a combination of PPMD and GPCC encoding was implemented. The results achieved were as good as the results achieved through 1D encoding. However, the compression and decompression speeds were significantly slower than those of the 1D encoders.

Table 33 summarizes the compression ratios and decompression speed in seconds of the three implemented methods in this thesis.

Sequences	Compression Ratios			Decompression Speed (s)		
	Proposed Method 1	Proposed Method 2	PPMD + GPCC	Proposed Method 1	Propose Method 2	PPMD + GPCC
Boxes_Rotation	5.5101	5.1859	4.1208	4.4431	5.3259	406.3319
Dynamic_Rotation	3.6509	3.6797	3.7584	4.2114	3.0624	395.0591
Outdoors_Running_1	3.8044	3.8003	3.7134	4.1633	4.9438	554.6839
Outdoors_Running_2	3.7277	3.7476	3.6598	3.6792	3.7042	472.008
Outdoors_Running_3	3.561	3.6006	3.5958	2.3873	2.2562	282.0022
Outdoors_Walking	3.4378	3.4769	3.4643	1.351	1.4761	128.3934
Poster_Rotation	5.3295	5.0331	4.2098	4.7007	5.3642	403.9383
Shapes_Rotation	3.3516	3.3976	3.977	1.1658	1.2746	80.3368
Slider_Close	4.3637	4.1936	3.7456	0.767	0.8412	23.26866
Urban	3.4325	3.4678	3.3657	1.3722	1.2924	89.899
Average:	5.5101	5.1859	4.1208	2.8241	2.9541	283.5921

Table 33. Comparison of compression ratios and decompression speeds of the 3 proposed methods

As can be seen from the table, the proposed method 1 or proposed method 2 can be used for compression according to the requirements. To achieve fast decompression, either method is suited. 3D compression can be used most efficiently for sequences with low event density.

REFERENCES

- [1] Mueggler, E., Rebecq, H., Gallego, G., Delbruck, T. and Scaramuzza, D., 2017. "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM". *The International Journal of Robotics Research*, 36(2), pp.142-149.
- [2] 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
- [3] [Online] Available: <https://youtu.be/bVVBTQ7I36I>
- [4] N. Khan, K. Iqbal and M. G. Martini, "Lossless Compression of Data From Static and Mobile Dynamic Vision Sensors-Performance and Trade-Offs," in *IEEE Access*, vol. 8, pp. 103149-103163, 2020
- [5] A. Y. Horita, R. Bonna, D. S. Loubach, I. Sander, and I. Söderquist, "Lempel-Ziv-Markov chain algorithm modeling using models of computation and For-syde," in *Proceedings of the Aerospace Technology Congress*, Oct. 2019.
- [6] B. Huang and T. Ebrahimi, "Event Data Stream Compression Based on Point Cloud Representation," *2023 IEEE International Conference on Image Processing (ICIP)*, Kuala Lumpur, Malaysia, 2023, pp. 3120-3124
- [7] M. Martini, J. Adhuran and N. Khan, "Lossless Compression of Neuromorphic Vision Sensor Data Based on Point Cloud Representation," in *IEEE Access*, vol. 10, pp. 121352-121364, 2022
- [8] J. Adhuran, N. Khan, and M. G. Martini, "Lossless Encoding of Time-Aggregated Neuromorphic Vision Sensor Data based on Point-Cloud Compression," in *Sensors*, vol. 24, no. 5, p. 1382, 2024.
- [9] Prophesee Event Camera [Online]. Available: <https://www.prophesee.ai/2024/02/27/metavision-image-deblur-production-ready/>
- [10] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco and T. Delbruck, "Retinomorphing Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output," in *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1470-1484, Oct. 2014
- [11] C. Brandli, R. Berner, M. Yang, S. -C. Liu and T. Delbruck, "A 240 × 180 130 dB 3 μs Latency Global Shutter Spatiotemporal Vision Sensor," in *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333-2341, Oct. 2014
- [12] C. Posch, D. Matolin and R. Wohlgenannt, "An Asynchronous Time-based Image Sensor," *2008 IEEE International Symposium on Circuits and Systems (IS-CAS)*, Seattle, WA, USA, 2008, pp. 2130-2133
- [13] G. Chen, H. Cao, J. Conradt, H. Tang, F. Rohrbein and A. Knoll, "Event-Based Neuromorphic Vision for Autonomous Driving: A Paradigm Shift for Bio-Inspired

- Visual Sensing and Perception," in *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 34-49, July 2020
- [14] W. Shariff, M. S. Dilmaghani, P. Kielty, M. Moustafa, J. Lemley and P. Corcoran, "Event Cameras in Automotive Sensing: A Review," in *IEEE Access*, vol. 12, pp. 51275-51306, 2024
- [15] Prophesee Event Camera. Accessed: Feb. 13, 2024. [Online]. Available: <https://www.prophesee.ai/event-based-sensors/>
- [16] Inivation Event Camera. Accessed: Feb. 13, 2024. [Online]. Available: <https://inivation.com/wp-content/uploads/2023/11/2023-11-iniVationdevices-Specifications.pdf>
- [17] Century Arks Event Camera. Accessed: Feb. 13, 2024. [Online]. Available: <https://centuryarks.com/en/silkyevcam-hd/>
- [18] Celepixel Event Camera. Accessed: Feb. 13, 2024. [Online]. Available: <https://www.celepixel.com/>
- [19] Samsung Event Camera. Accessed: Feb. 13, 2024. [Online]. Available: https://rpg.ifi.uzh.ch/docs/CVPR19workshop/CVPRW19_Eric_Ryu_Samsung.pdf
- [20] Insightness Event Camera. Accessed: Feb. 13, 2024. [Online]. Available: <https://www.insightness.com/technology/>
- [21] K. Mammou, P. A. Chou, D. Flynn, M. Krivokuća, O. Nakagami, and T. Sugio. G-PCC Codec Description v2, Standard ISO/IEC JTC1/SC29/WG11 N18189, 2019.
- [22] G-PCC codec description v12, Standard ISO/IEC JTC 1/SC 29/WG 7 N20626, 2021
- [23] J. Cleary and I. Witten, "Data Compression Using Adaptive Coding and Partial String Matching," in *IEEE Transactions on Communications*, vol. 32, no. 4, pp. 396-402, April 1984
- [24] N. Khan, K. Iqbal and M. G. Martini, "Time-Aggregation-Based Lossless Video Encoding for Neuromorphic Vision Sensor Data," in *IEEE Internet of Things Journal*, vol. 8, no. 1, pp. 596-609, 1 Jan.1, 2021
- [25] 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
- [26] M. Burrows and D. J. Wheeler, "A Block-Sorting Lossless Data Compression Algorithm," *Digital Equipment Corporation, SRC Research Report 124*, 1994
- [27] A. Z. Zhu, L. Yuan, K. Chaney and K. Daniilidis, "Live Demonstration: 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

- [28] E. Perot, P. De Tournemire, D. Nitti, J. Masci, and A. Sironi, "Learning to Detect Objects with a 1 megapixel Event Camera," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 16639-16652
- [29] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," in *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098-1101, Sept. 1952.
- [30] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520-540, Jun. 1987.
- [31] J. Alakuijala and Z. Szabadka, "Brotli Compressed Data Format," *Internet Engineering Task Force*, RFC 7932, 2016.
- [32] I. Pavlov. LZMA SDK (Software Development Kit). [Online]. Available: <https://www.7-zip.org/sdk.html>
- [33] T. Nishimoto and Y. Tabei, "LZRR: LZ77 Parsing with Right Reference," 2019 *Data Compression Conference (DCC)*, Snowbird, UT, USA, 2019, pp. 211-220.
- [34] Y. Collet and E. M. Kucherawy. (Jul. 2018). Zstandard-Real-Time Data Compression Algorithm. [Online]. Available: <http://facebook.github.io/zstd/>
- [35] P. Deutsch and J.-L. Gailly, "Zlib Compressed Data Format Specification Version 3.3," RFC 1950, May 1996.
- [36] V. N. Anh and A. Moffat, "Index compression using 64-bit words," *Software: Practice and Experience*, vol. 40, no. 2, pp. 131-147, 2010.
- [37] D. Lemire and L. Boytsov, "Decoding billions of integers per second through vectorization," *Software: Practice and Experience*, vol. 45, no. 1, pp. 1-29, Jan. 2015.
- [38] S. H. Gunderson. (Apr. 2015). Snappy: A Fast Compressor/Decompressor. [Online]. Available: <https://github.com/google/snappy>
- [39] D. Blalock, S. Madden, and J. Guttag, "Sprintz: Time series compression for the Internet of Things," in *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 3, p. 93, 2018.