MOTION COMPENSATED COMPRESSION FOR EVENT-BASED CAMERAS

ARNOB KUMAR BAIRAGI
Bachelor of Science, Rajshahi University of Engineering and Technology, 2019

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

MOTION COMPENSATED COMPRESSION FOR EVENT-BASED CAMERAS

ARNOB KUMAR BAIRAGI

Date of Defence: November 25, 2022

Dr. H. Cheng Supervisor	Associate Professor	Ph.D.
Dr. J. Zhang Committee Member	Associate Professor	Ph.D.
Dr. J. Anvik Committee Member	Associate Professor	Ph.D.
Dr. W. Osborn Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

I dedicate this thesis to my beloved parents, who have trusted me.

Abstract

Event cameras detect significant changes at each pixel asynchronously and report these events in real-time. A changing scene can generate many events in a short time. Efficient storage and transmission are necessary for further processing of this event data. Inspired by this necessity, we propose a lossless Motion Compensated Compression algorithm based on Optical Flow (MCCOF) for event cameras. We analyzed our proposed algorithm performance compared with the lossless spike coding algorithm. We found that our MCCOF algorithm achieves a higher compression ratio on most datasets compared to the spike coding algorithm. Using a real-time event-based optical flow algorithm for motion compensation, our algorithm does not significantly increase the computational time for compression and decompression.

Acknowledgments

I want to thank my parents and God for helping me get to where I am now. After that, my deep gratitude goes first to Dr. Howard Cheng, to whom I cannot express my honor by only writing this acknowledgment. I appreciate him for giving me this amazing opportunity to fulfill one of my dreams. I want to thank him for his help with the admission process and for acting as my academic advisor by being available whenever I needed any advice on my research, classes, or anything else. It is an immense privilege and honour to have had his guidance throughout my master's thesis.

I want to thank the other committee members, Dr. John Zhang and Dr. John Anvik, for their advice and assistance throughout my research. Also, I want to thank all the professors and staff members in the Department of Mathematics and Computer Science.

I am grateful for all the special people I have met in Lethbridge, Omee Apu, Mouwa Apu, and Himel Vaiya. They supported me in Lethbridge, welcomed me into their home, and made me feel like I was in my own country with my relatives.

Finally, I wish to show appreciation for my parents, my better half. Also, Jetthu, Boroma, Subroto Dadu, Joya Didima, and everyone supported me in familiarizing me with this new country and environment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Event-based cameras are one type of imaging sensors that can respond to significant local changes in log-luminance, called events. There are some differences between event-based cameras and conventional frame-based cameras. Conventional cameras capture and store videos in the form of frames, and data is captured even if there is no change in the scene. However, event-based cameras do not specifically output the intensity of each pixel as frame cameras do. Each pixel compares the current and previous log-luminance levels and fires an event when the difference exceeds a user-defined positive or negative threshold. These events can arrive asynchronously as event-based cameras only report significant log-luminance changes, which may occur only in some pixels.

The Davis Dynamic Vision Sensor (DVS) is a biologically motivated event-based camera. It is used for modern computer vision applications, including high-frame-rate video reconstruction with high dynamic range (HDR), high resolution and 3D reconstruction of human motion, motion classification, and tracking [16]. As event-based cameras respond only when significant log-luminance changes happen, a moving object can generate many events in a short time based on surrounding conditions. For example, a falling water drop may generate nearly 11.5 million events over 6 seconds, with about 7.7 million bytes in a second [31]. Transmission and sometimes storage of these large number of events are required in addition to any processing for the computer vision tasks at hand. Therefore, for efficient storage and transmission, efficient compression of these events has become an emerging research topic for event-based cameras.

Events are mainly represented in a raw format using the Address Event Representation (AER) protocol [22]. For example, the AEDAT 4.0 format uses 96-bit representation for each event, and its earlier version AEDAT 3.1 uses 64-bit representation. Each event is represented by four attributes, including the spatial location (x, y) and timestamp (t) of each event, with a polarity $(p = \pm)$ representing the log-luminance increase or decrease. The main challenge for compressing events is the unique event data stream properties. When events are generated from a DVS camera, they are asynchronously fired as event tuples (x, y, t, p). Since the events are generated as soon as significant changes are detected in a pixel, there may be little spatial structure among consecutive events. This lack of spatial structure makes event compression more challenging as most conventional image and video compression algorithms rely on spatial structures to remove redundancies.

Compression algorithms can generally be classified into two types: lossless compression and lossy compression. In lossless compression, the original data is reconstructed perfectly from the compressed data. On the contrary, lossy compression only permits the reconstruction of an approximation of the original data, allowing a larger reduction in storage.

Several algorithms have been developed in the past. Bi et al. [5] first proposed a lossless predictive compression technique that could compress events efficiently. They modified their algorithm with inter-cube prediction utilizing the temporal correlation among macro-cubes [11]. Khan et al. [25] proposed a time aggregation-based lossless compression algorithm, which showed a comparatively better compression ratio (CR) than the algorithm by Bi et al. [5]. Data compression algorithms not specifically designed for DVS events can also be used [37].

In this thesis, we utilized the event-based optical flow algorithm proposed by Ridwan and Cheng [34, 35] to predict current events from previous events based on motion. We designed and implemented a lossless Motion Compensated Compression algorithm based on Optical Flow (MCCOF). We used PKU-DVS datasets from Peking University [31] for

comparing our results to previous works. Our proposed MCCOF algorithm shows a better compression ratio (CR) than the spike coding algorithm [5] for most scenes.

1.1 Main Contributions

In this thesis, we proposed a lossless Motion Compensated Compression algorithm based on Optical Flow (MCCOF). It is an improved version of Bi et al. [5] spike coding algorithm with motion prediction. This motion compensation is done with the help of Ridwan and Cheng's optical flow algorithm [35]. To our knowledge, it is the first work using optical flow for motion compensation.

It is shown through experiments that the proposed algorithm achieves higher compression ratio on many scenes compared to the spike coding algorithm. This is achieved by incorporating motion compensation that is used selectively in some situations. Our algorithm adaptively decides whether motion compensation is used.

The event-based optical flow algorithm by Ridwan and Cheng [35] can be applied in real-time and has a very low computational cost (about 2μs per event). Using an event-based optical flow algorithm for motion compensation does not significantly increase the computational time for compression, unlike many other traditional motion compensation algorithms which often require exhaustive search.

1.2 Organization of the Thesis

In Chapter 2, background concepts are introduced. In Chapter 3, previous approaches for data compression for event-based cameras are described. Chapter 4 presents the proposed lossless Motion Compensated Compression algorithm based on Optical Flow (MCCOF). In Chapter 5, we show the experimental results comparing MCCOF against previous works to demonstrate the effectiveness of the proposed algorithm. Concluding remarks and possible directions for future work are given in Chapter 6.

Chapter 2

Background

In this chapter, some background information and definitions are given. First, conventional frame-based cameras are reviewed. Next, the event-based cameras are described and compared to conventional frame-based cameras. The topic of data compression is introduced, and specific compression methods such as predictive coding, arithmetic coding, and context modelling are explained. A brief overview of some of the previous compression algorithms for event-based cameras is given. Special attention is given to the spike coding algorithm [5] as it is the inspiration of the algorithm proposed in this thesis. Finally, the event-based optical flow algorithm used in our work is reviewed.

2.1 Digital Cameras

2.1.1 Conventional Frame-based Cameras

A video is represented by successive image frames. Each image frame contains an image captured by an array of imaging sensors. In this way, conventional cameras record and represent videos. These frame-based cameras have a specific frame rate (the number of images they can capture per second). Typically these cameras have frame rates from 20–40 frames per second. Even if the scene is unchanged or few changes have occurred, these image frames are captured and stored. Figure 2.1 shows an example of a frame from a conventional camera. Therefore, this process generates redundant data in many cases. Dealing with this redundant data requires more computation and electrical power. Also, additional compression techniques are often used to deal with these redundant data.

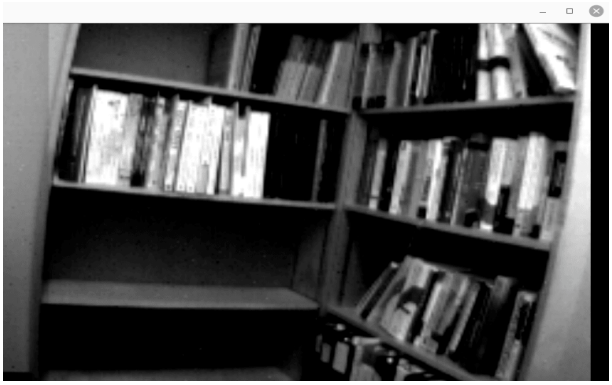

Figure 2.1: Conventional frame-based camera captured image.

With conventional frame-based cameras, there is some delay between the time the scene is changed and the time the scene is captured and available for processing, as these cameras must send captured frames synchronously at regular time intervals. This delay is called latency, and it is important as it is the delay between what is happening in front of the camera sensors and what is processed. For example, though broadcast television has a typically high latency, this higher latency may be an acceptable trade-off to achieve better video quality [39]. However, high latency cameras are not suitable in many situations, such as monitoring a restricted area or sensitive robotics applications. These frame-based cameras are mostly used because of their low cost and wide availability in local markets. Many algorithms have been developed for these commonly used cameras, but there are still some drawbacks associated with the applications of these cameras.

For applications in robotics and autonomous vehicles, a fast response to the environment is required. Therefore low-latency cameras are required. Since some changes may happen between two successive frames, motions at a higher speed than the frame rate will be blurred. Algorithms designed for these types of cameras cannot detect changes faster than the frame rate. This is also called aliasing, and it happens when the frequency of changes is more than half of the frame rate [17].

For conventional cameras, when the range of intensity in the scene is high, some details are lost. Most cameras cannot simultaneously capture very bright or very dark objects with

Figure 2.2: Example of low dynamic range in conventional camera.

sufficient details. Typically, additional settings such as lens' aperture and exposure time are applied to control the amount of light that reaches the camera sensors. If there are very bright or dark objects in the same scene, the camera settings must be tuned to allow the bright or dark, but not both, objects to be seen. That is why these conventional cameras have a low dynamic range. An example is given in Figure 2.2. Here, the image's brightness is decreased in some areas to allow the sky to be seen in detail. The brightness of the street light has been decreased, allowing it to show the features of that portion. However, High Dynamic Range (HDR) techniques can be used to obtain details across intensity ranges [10]. Using this technique requires taking images at multiple exposure settings and then solving some optimization problem to compute the radiance at each pixel. This results in increased computational complexity and may not work for rapidly changing scenes.

In summary, conventional cameras have low-dynamic range and high latency for certain applications. They also use a large amount of memory or disk storage to deal with this additional redundant data, which wastes processing time and increases energy consumption.

2.1.2 Event-based Cameras

Event-based cameras are biologically motivated camera used for modern computer vision applications, including high-frame-rate video reconstruction with high-dynamic-range (HDR), and high resolution and 3D reconstruction of human motion and tracking [16].

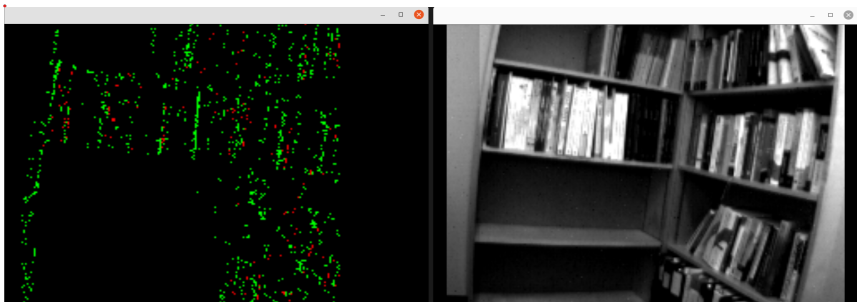

Figure 2.3: Event-based camera output (left) and frame-based camera output (right).

They are inspired by the retinal structure of the human eye. Unlike conventional image sensors in which image data are transmitted synchronously, event-based cameras are asynchronous devices that transmit only events associated with significant changes in pixel log-luminance when a large enough change is detected and each pixel can be triggered independently [16].

Events are represented by the polarity (p), coordinates of the pixel's location (x,y), and timestamp (t) of those events. Besides the polarity, coordinate, and timestamp, there might be other information such as camera motion reported by the camera. However, in this thesis, we are only concerned with polarity events. Polarity is the direction of significant changes in log-luminance in a scene. If there is a significant change in log-luminance, the camera detects it and generates an event. However, if there are no significant changes, the camera does not produce any output as this camera only responds when there are significant changes in the scene. The output of an event-based camera, together with the frame-based camera output, are shown in Figure 2.3. On the right is a representation of the events in the DVS captured for a window of time. The green pixels are positive (from dark to bright) events, and the red are negative events.

Due to low latency, event cameras are beneficial for robotic navigation and other applications. For example, in robotics or autonomous situations, the device needs to make decisions by observing its surrounding situations. So, it needs a camera which can respond

quickly. Event-based cameras perform better than conventional cameras as they require less power, fewer computations, and quicker reaction times [21].

These cameras have a higher dynamic range than conventional cameras since they depend on changing log-luminance levels rather than their absolute values. Also, each pixel can operate independently of each other, so even if one pixel is very bright and one is very dark, the camera can detect significant changes independent of each other. In event-based cameras, each pixel generates events independently. The continuous-time photoreceptor output that encodes luminance logarithmically is constantly monitored to determine whether there have been any significant changes since the last event. When a change in log-luminance exceeds a threshold value, an event is triggered. After the event is generated, the pixel is reset to memorize the new log-luminance value [16].

However, working with this technology is sometimes difficult since only the directions of changes are reported, but the magnitudes are not. Several works have been done to reconstruct the intensity frames from camera data [3, 24, 26]. Through these methods, it is possible to reconstruct the original video. However, the reconstructed video is only a rough approximation that requires more calculations and numerous camera movements to compute.

2.1.3 The Davis Dynamic Vision Sensor

Event-based cameras have low latency (about 10 µs on the lab bench and sub-millisecond in the real world), high speed, and high dynamic range ($> 120dB$) [16]. There are some commercial event based cameras, DVS128, DAVIS240, DAVIS346 produced by iniVation that have differences in their resolutions [16]. We refer to these cameras as Dynamic Vision Sensor (DVS). For DVS cameras (Figure 2.4), several communication protocols or storage formats are used to represent event data efficiently. Address Event Representation (AER) is a communication protocol that is designed to communicate sparse neural events between neuromorphic chips [7]. These events can occur asynchronously to respond to stimulus,

Figure 2.4: The DAVIS 240*B*.

and there may be no events generated for a long time if there is no stimulus. The AER is an asynchronous transmission protocol.

When an event is generated from the DVS, pixels coordinates (x, y) and other information such as the polarity (p) and timestamps (t) are sent through the AER bus. This mechanism makes the neuromorphic chip act similarly to the human brain in a large-range communication point of view [7]. The Java Address-Based Representation (jAER) and C Address Event Representation (cAER) libraries are two libraries available for accessing the DVS output in AER format [7, 23].

Sometimes, it is necessary to store DVS output events for further use. The AEDAT (Address Event Data) format is used to store DVS events streams [22]. For example, the AEDAT 4.0 format uses 96-bit representation for each event, and its earlier version AEDAT 3.1, uses 64-bit representation. Efficient compression of these events has become an important research topic for event-based cameras for efficient storage and transmission.

2.1.4 Summary of Comparison

The main difference between event-based cameras and conventional frame-based cameras is that conventional cameras capture and store videos in frames at regular intervals, and data is captured even if there is no change in the scene. On the other hand, event-based cameras do not output the intensity signals as frame-based cameras do. Each pixel compares

the current and previous log-luminance levels and generates an event when the difference exceeds a threshold. These events can arrive asynchronously as event cameras only report significant log-luminance changes, which may occur only in some pixels.

As event-based cameras respond only when significant log-luminance changes happen, moving objects can generate many events in a short period of time based on surrounding conditions. However, a significant problem with these event-based cameras is that a rapidly changing scene may generate a large amount of output. As a result, it is beneficial to implement some compression algorithms for the storage and transmission of these events.

2.2 Data Compression

When images, videos, event streams, or other data need to be transmitted or stored, an encoding is needed to represent this information. A sequence of symbols can represent the data, and each symbol is drawn from a set of possible symbols. The frequency (or probability) of each type of symbol can be used to compute a quantity known as the entropy representing the amount of information inherent in the data [17]. Suppose that the possible symbols are $s_1, \ldots, s_n$ with probability $p_1, \ldots, p_n$, respectively. Then the entropy is defined as

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i. \tag{2.1}$$

The amount of data used to represent an information source depends on the encoding chosen. For example, a grayscale image can be encoded as a sequence of intensity values of each pixel. A DVS event stream can be encoded as a sequence of the binary representations of each event consisting of the tuple (t, x, y, p).

In data compression, one attempts to find an encoding method so that the length of the encoding is shorter than the original encoding. If the length of the original encoding is A and the length of the compressed encoding is B, then the compression ratio is defined as

$$CR = \frac{A}{B}. \tag{2.2}$$

Another measure of compression performance is the bit rate, which is the number of bits used to encode each symbol [17].

For lossless compression, it is important that the exact original encoding can be obtained from the compressed data. For lossy compression, further compression can be achieved by requiring only a "close" approximation of the original encoding be recovered. For this thesis, we will focus only on lossless compression.

Data compression is typically accomplished by reducing the redundancy that exists in the original encoding of the data. For general image and video data, there are three common types of redundancies that are considered—coding redundancy, spatial redundancy, and temporal redundancy [17].

Coding redundancy exists when the average number of bits used to encode each symbol is higher than the entropy. The entropy is highest when the probabilities for all symbols are the same. But when the probability distribution is highly skewed, the entropy is significantly lower. In this case, there is coding redundancy and a different encoding can be chosen to lower this redundancy. Coding redundancy is reduced by using an entropy or symbol encoder. Common entropy encoders are Huffman coding [20] and arithmetic coding [36].

Spatial and temporal redundancies exist because symbols that are located spatially or temporally close to each other are similar. For example, the intensity of an image pixel is likely to be similar to those around it spatially. Spatial and temporal redundancies are typically reduced by a mapper, which changes the encoding to produce a highly skewed probability distribution in the new encoding. This results in a lower entropy, so that compression can be achieved by an entropy encoder. Typical mappers include transforms such as discrete cosine transform [33] or wavelet transform [9], as well as various types of predictive coding [14, 15] and motion compensation [40].

In data compression of image and video data, the first step is to apply a mapper to reduce spatial and temporal redundancies. The result is then compressed by an entropy encoder. To decompress, an entropy decoder is applied, followed by an inverse mapper.

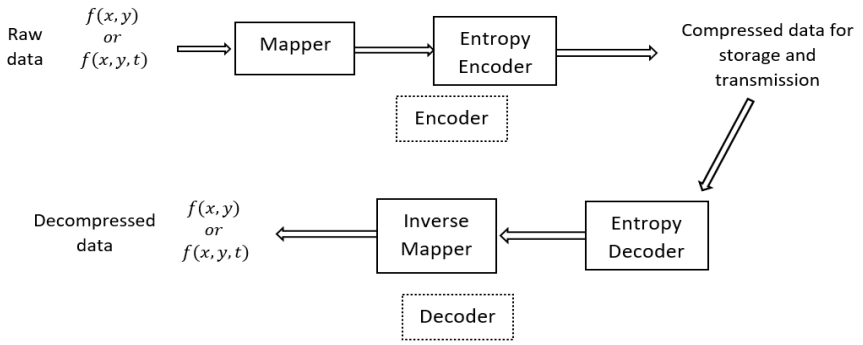

Figure 2.5: Functional block diagram for image and video compression and decompression.

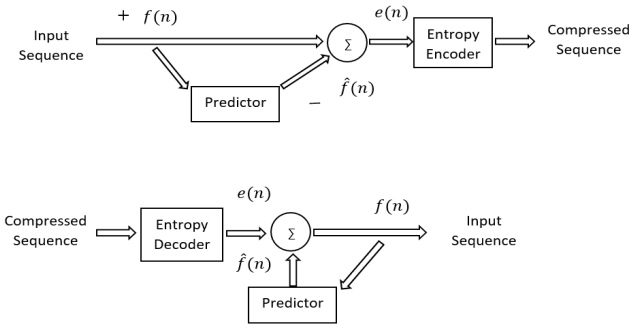

Figure 2.6: Block diagram for compression and decompression using predictive encoding.

This is illustrated in the block diagram in Figure 2.5. This is a standard model for the various components of a lossless compression algorithm [17].

2.2.1 Predictive Encoding

Predictive coding is based on reducing the redundancies among pixels that are close to each other spatially and temporally. This is done by predicting the current value from previously encoded values, and encoding only the prediction error. To encode the first value, 0 is used as the prediction. If the predictor is accurate, the errors are likely close to 0. For example, one simple predictor is simply to predict the current value as the same as the previous value. Predictive encoding is illustrated in Figure 2.6.

From here we can observe the essential components for a lossless predictive coding sys-

tem. It consists of an encoder and decoder, each of them containing an identical predictor. The data to encode is a sequence of values $f(n)$ for $n = 1,2,3,\ldots$ As successive values enter the encoder, the predictor generates the predicted value of each sample based on a specified number of past samples. The output of the predictor, denoted with $\hat{f}(n)$ is used to form the difference or prediction error.

$$e(n) = f(n) - \hat{f}(n). \tag{2.3}$$

This error is encoded using an entropy encoder. This prediction-based compression makes the overall compression of the data stream more efficient as it reduces the redundancies in data by predicting with nearest events, and the resulting data will have a probability distribution highly skewed and centred around zero. On the decoder side, the inverse operation is completed to decompress or recreate the original input sequence.

$$f(n) = e(n) + \hat{f}(n). \tag{2.4}$$

2.2.2 Arithmetic Coding

Arithmetic coding is a type of entropy coder. It is the process of encoding information using an average number of bits close to the entropy. The algorithm receives a stream of input symbols and replaces them with a single floating-point number in $[0,1)$. Using the symbol probabilities given by the model, successive message symbols reduce the interval size. The more likely symbols reduce the range by less than the unlikely symbols and add fewer bits to the message. Comparatively more bits are needed to represent the output number for more unlikely input data.

Figure 2.7 represents the coding of the message $a_1a_2a_3a_4$. The message is assumed to contain the entire half-open interval $[0,1)$ at the start of the coding procedure. Based on the probability of each source symbol, the intervals are initially separated into four areas. Subinterval $[0,0.1)$, for example, is connected with symbol a_1. After the first symbol a_1

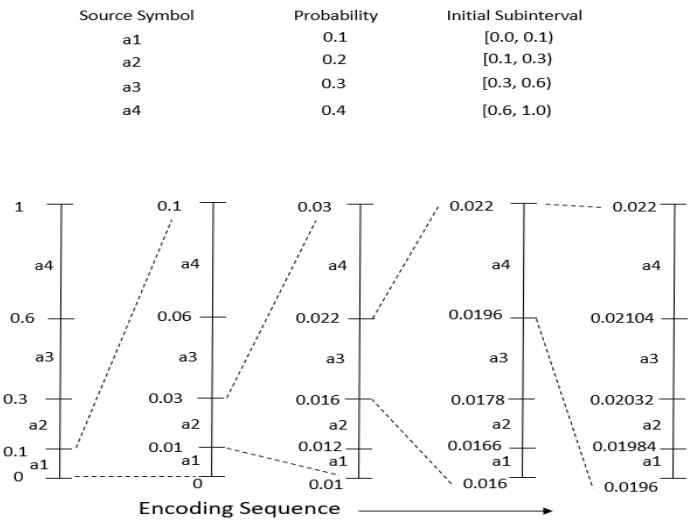

Figure 2.7: Arithmetic coding procedure (for the message $a_1a_2a_3a_4$).

is encoded, the range is now $[0, 0.1)$. As a result, it is extended to fill the Figure 2.7's full height, with the values of the reduced range indicating the range's boundary. The reduced range is then subdivided based on the probability of the source symbol, and the process is repeated for the next message symbol. Symbol a_2 reduces the subinterval into $[0.01, 0.03)$, a_3 reduces it even further into $[0.016, 0.022)$, and so on. The range is reduced to $[0.0196, 0.022)$ with the last message symbol a_4, which can be assigned as a particular end-of-message indication. The message is encoded as any value in the final range. For example, 0.02 can be used as an encoding of the message.

An advantage of arithmetic coding is separating the coding from the probability model of the symbols. This change makes it possible for arithmetic coding to use a probability model that changes to adapt to the data being encoded.

2.2.3 Context Modelling

In a data stream, many different types of information are encoded, such as the intensity, coordinates, prediction errors, timestamps, polarity, etc. Each type of information has different sets of symbols with different probabilities. Therefore, it is more accurate to estimate the probabilities of each type of information separately. This is known as context modelling,

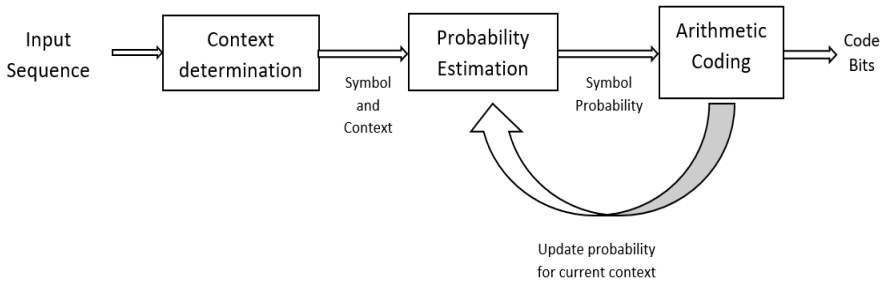

Figure 2.8: Adaptive context-based arithmetic coding approach.

which means that there is a different probability model for each type of information. The type of information is known as the context.

For image and video input, we can define the context of the current pixel based on previous surrounding pixels. Specific context types need to be set before encoding and decoding. As long as the context is defined by the data already encoded, both encoder and decoder can compute the same context at the next step. There is a balance between having enough contexts to allow probability models to be accurate for different situations and too many contexts so that there is not enough data for the probability models.

An adaptive, context-dependent probability model is used to increase the accuracy of the probabilities employed. Adaptive probability models update symbol probabilities as symbols are encoded or become known. Thus the probabilities adapt to the local statistics of the encoded symbols, illustrated in Figure 2.8.

As each symbol enters the encoding process, the algorithm determines the correct context to use. The context determines the correct probability model to use to encode using arithmetic coding. After the symbol is encoded, it is used to update the probability model to adapt to the statistics of the symbols being encoded.

For example, an image compression algorithm can define the context of a pixel being encoded by a neighbourhood around that pixel. A context can be computed based on the pixel values in the neighbourhood.

Chapter 3

Previous Works

3.1 DVS Specific Spike Coding

Bi et al. [5] proposed a lossless predictive compression technique for DVS event cameras. The event camera produces a sequence of events (also called spikes) as the input to the compression algorithm. The event sequence is partitioned into multiple macro-cubes by time, and each of the macro-cubes has the full spatial resolution of the pixel array using an adaptive macro-cube partitioning strategy. The spike locations and polarities in each macro-cube are encoded using their proposed strategy, and entropy encoding is applied as the final step. The overall procedure is illustrated in Figure 3.1.

Adaptive Macro-cube Partitioning

Spikes are partitioned into macro-cubes to smooth the fluctuation of the number of spikes in each macro-cube, and limit the context size for the entropy coder. Each macro-

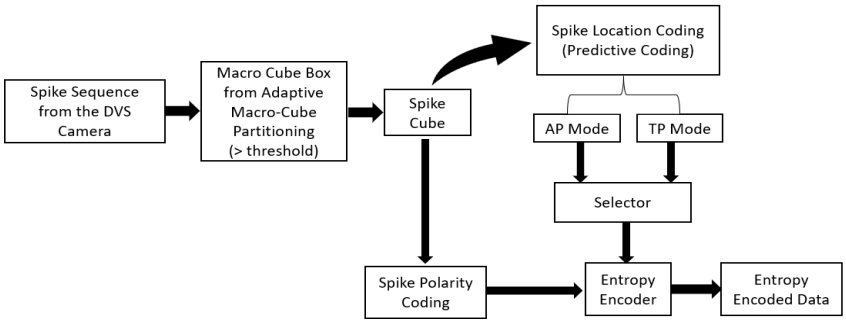

Figure 3.1: Overall procedure for spike coding algorithm proposed by Bi et al. [5].

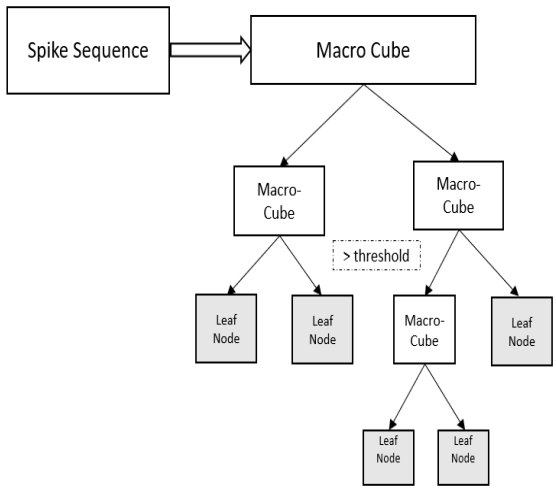

Figure 3.2: Adaptive macro-cube partitioning strategy.

cube is then spatially partitioned into smaller spike-cubes based on the spatial locations of the spikes. A macro-cube with a maximum time length is used as the root of a binary tree for this partitioning. From the starting of the root, if the total number of spikes in each node is bigger than a predefined threshold, the node is equally split into two small macro-cubes recursively, and the leaf nodes of the binary tree contain the macro-cubes. This procedure is illustrated in Figure 3.2. The macro-cube is subsequently divided into smaller spike-cubes. The spike-cube encoding technique consists of spike location coding and spike polarity coding.

Location coding algorithms are applied to the events in each spike-cube. They proposed two methods for location coding in a spike-cube. These are called address-prior (AP) and time-prior (TP) modes. Each method is applied by the algorithm, and the one with the lowest bit rate is selected for each spike cube.

Address-Prior (AP) Mode

Address-prior mode is designed for spatial-decentralized spike-cubes, using one type of predictive coding. A spike-cube is spatial-decentralized if the events in the spike-cube are not spatially clustered. The operation of this mode is illustrated in Figure 3.3.

The operation can be described by a number of steps:

1. By projecting all the spikes in a spike-cube to the *xy* plane, a location histogram is created, such that the number of spikes in each location is stored.

2. This histogram is represented by two separate maps. First, a binary map is used to indicate whether a location has zero or non-zero number of spikes. A second map is used to store the number of spikes in each location.

3. The binary map is encoded using a context-based arithmetic coder [38]. The content of the binary map in the surrounding neighbourhood is used as the context in this encoding.

4. The sequence of non-zero values in the second map is encoded by the entropy encoder.

5. The time differences between the consecutive spikes for a particular location are encoded by the entropy encoder.

In general, Bi et al. [5] measured the time intervals between consecutive spikes for a specific pixel to predict the occurring time of the subsequent spike. Later they introduced Referenced Address-Prior mode [11]. The spikes would find their nearest references according to the timestamps in the corresponding spike train after encoding the location histogram map and histogram counts and encoding the reference count. The timestamp was converted to the time difference between a spike and its reference spike, which is then entropy encoded.

Time Prior (TP) Mode

This mode is used for centralized spike-cubes, and it is also a type of predictive coding. The operation can be described as a number of steps:

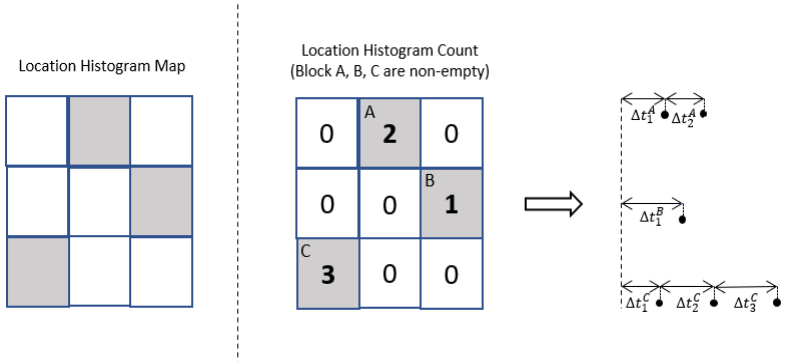

Figure 3.3: Address-Prior (AP) mode strategy.

1. All the spikes are projected to its time axis appearing in a timeline by finding a centre point (x_c^*, y_c^*). The centre point is,

$$(x_c^*, y_c^*) = argmin_{x_c, y_c} \sum_{i=1}^{n} |x_i - x_c| + |y_i - y_c| \tag{3.1}$$

   Where (x_i, y_i) denotes the spike locations $(i = 1, 2, ..., n)$.

2. The centre point is encoded directly or differentially by referencing the centre point $(x_{c,prev}, y_{c,prev})$ of the previous collocated spike cube.

3. The timestamp Δt of events are differentially encoded, and the motion vector $\Delta x_i, \Delta y_i$ passed into the entropy coder (CABAC) directly [29]. This is illustrated in Figure 3.4.

In [11], they improved their algorithm by finding the nearest reference spikes by comparing the timestamps. The time differences between the spikes and the corresponding reference spikes are encoded along with the reference counts.

Spike Polarity Coding

Bi et al. [5] used the intensity property of light to encode the events polarity (p) because polarity depends on light intensity and this intensity usually spreads over the surrounding areas. So, it can be considered that if the previous spike/event is "+ (On)" (or "− (Off))"

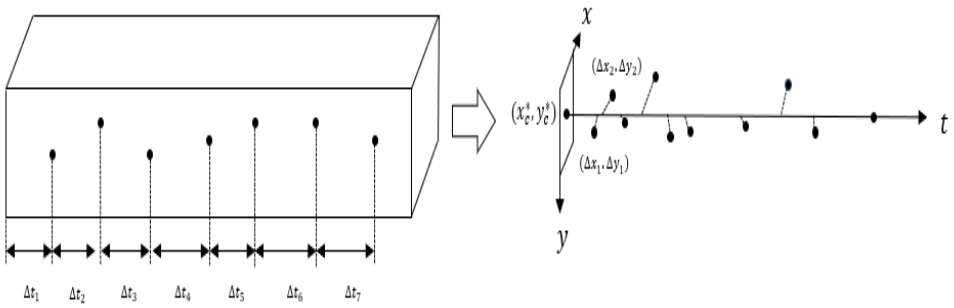

Figure 3.4: Time-Prior (TP) mode strategy.

polarity, then there is a high probability that the next spike will have the same polarity of "+" (or "−"). Thus, the encoding of the spike polarity exploits the previous spike polarity as the context for the current one and passes into the entropy encoder. However, this may not be accurate for scenes where maximum intensity or brightness appears. In practice, assuming that the intensity will be the same for the surrounding areas usually leads to good results.

Similar to Referenced Address-Prior (RAP) and Time-Prior (RTP) mode Dong et al. [11] also used previous reference spike polarity as a context for the current spike polarity in their further work.

3.1.1 Time Aggregation Based Encoding

Khan et al. [25] proposed a time aggregation-based lossless video encoding technique for neuromorphic vision sensor data. This technique uses temporal data aggregation arrangements in a specific format for lossless video encoding techniques to achieve high compression ratios. The event stream is converted into a video-like format, converting the DVS spike event sequence into synchronized video frames with strong spatial and temporal correlation by projecting the DVS spike event stream into a series of frames, each with

the full resolution of the pixel array. They proposed to divide the spike sequence into two separate frames, one associated with the positive increase of log luminance and one for the decrease of log luminance. If the preceding event's intensity is positive (or "+") on a pixel, the polarity of the next event will almost certainly be positive (or "+") on the same pixel. They proposed the merging of the frames of each polarity with the same timestamp into one single superframe consisting of the "positive polarity" frame on the left and the "negative polarity" frame on the right. "Superframes" are used to arrange data so that inter-frame correlation is increased. Next, video encoding was performed, with each superframe represented as a video frame. All of the spike sequence information is contained in the superframe. Their strategy employs the lossless compression mode of recent video encoding standards. They obtained a comparatively better compression ratio (typically within 3.5 to 4.9) because of multiple steps though this ratio depends mainly on the datasets.

3.1.2 Compression algorithms adapted to DVS data

Entropy Encoding: Entropy encoding is a process in which codes are assigned to symbols to match code lengths to symbol probabilities. They are helpful in a variety of DVS applications [16]. The most common entropy coding schemes are Huffman coding and Arithmetic coding, which may be applied directly to DVS data by treating each field of the event as an input symbol. Because these strategies have modest compression advantages as a single compression strategy, entropy encoders have been used as the last step in many advanced coding approaches. Its compression ratio remains within 1.8 to 1.9 [25] when it is applied directly to the DVS data.

Dictionary-based compression: Dictionary coding schemes work by replacing large strings with shorter codewords. Variable-length symbols are encoded as single tokens, and each token serves as an index to a phrase dictionary. If the tokens are smaller than the phrases, they will be replaced, resulting in compression. Some advanced dictionary coders, such as Zstd [8], LZMA [32], and Brotli [1], use multi-level encoding and entropy coding

to compress their dictionary codewords further. By transforming events into a multivariate stream of integers and using the dictionary-based substitution concept to the DVS data, dictionary-based compression can be implemented. The compression gains increase as the frequency of repeated integers increases.

These compression techniques can significantly reduce the quantity of the data, which can help alleviate the data rate and storage constraints of DVS surveillance-related applications [28]. However, the lack of repetitive patterns in the asynchronous stream of DVS data can limit the compression gains of the dictionary-based compression strategies. Its compression ratio remains within 2.8 to 4.9.

Internet of Things (IoT) specific compression: Blalock et al. [6] proposed a compression strategy named Sprintz for resource-constrained devices. Its goal was to achieve state-of-art compression gain without violating the latency constraints of the IoT devices. SprintzFIRE, SprintzDelta, and SprintzFIRE+Huf are the three main variants of Sprintz strategy. SprintzFIRE is based on a forecast called Fast Integer Regression (FIRE). This algorithm predicts the current sample based on information on the previous samples. The prediction errors are usually Huffman encoded. However, SPrintzDelta skips the Huffman coding step and replaces the FIRE algorithm with delta coding to achieve higher compression speed. Moreover, the third variant, SprintzFIRE+Huf, achieves a trade-off between compression ratio and compression speed by employing a joint combination between FIRE and Huffman encoding. The Sprintz algorithms converts the event stream into a multivariate time series of integers. Techniques for time series compression are applied to take advantage of the properties of the event stream as a time series. It can be helpful in scenarios where higher compression gains with reduced power consumption have essential performance criteria [16]. Its compression ratio remains within 2.5 to 3.7.

Fast Integer Compression: These compression strategies are used in real-world situations where rapid compression is required. These fast integer compression algorithms are specifically designed to encode or decode billions of integer arrays in search engines

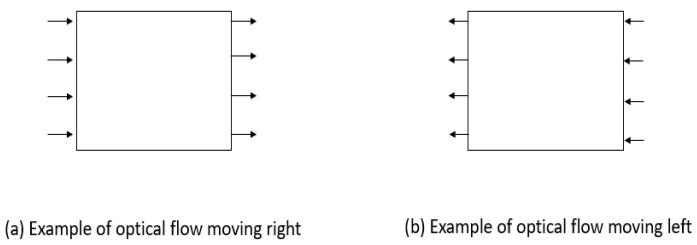

Figure 3.5: Examples of optical flow.

and relational database applications. For fast integer compression algorithms, the most commonly used are Samole8B [6], Memcpy [6], SIMD-BP128 [27], and SNAPPY [18]. These strategies could be used in scenarios where DVS data is transported to cloud storage and computing servers for visual data processing such as an event, action, person or object recognition/classification, and context awareness [30]. These techniques are applied to DVS data by converting events into a column major format (vector or integer). Its compression ratio remains within 1.3 to 3.0.

3.2 Optical Flow for Event-based Camera

In our daily lives, analyzing motion of objects plays a crucial part. It helps us in different ways. As an example, when we try to cross a street safely, we need to analyze the traffic situations on both sides. This motion detection requires a change in the position of an object related to its surrounding environment. This movement between a perceiver and objects in a scene generates visible motions of objects, surfaces, and edges. This term is called the optical flow. In computer vision, optical flow is the pattern of apparent motion indicated by a vector for each pixel in a scene (Figure 3.5).

Optical flow has many computers vision and robotics applications such as navigation control, motion detection, and classification. Optical flow can be used to detect movement in a scene. Many optical flow algorithms have been developed for conventional frame-based cameras [4, 12, 19]. Calculus principles, such as partial derivatives, are frequently

used in these methods. SpikeNet Technology has created a processing algorithm [13] based on neurobiological principles that are efficient and effective. This algorithm is capable of calculating optical flow in real-time. The system can offer a direction of motion in 8 or 16 potential directions at each place in the image and a number for the distance (in pixels) between two successive frames. A dense motion map can be created when thresholds are set at shallow levels, with each pixel generating a motion value. The noise can be controlled by changing the threshold value. Although the speed of this algorithm is its major benefit, this system is only meant for conventional cameras.

Dramas et al. [12] proposed a neurobiological algorithm that calculates optical flow in real-time. This algorithm uses thresholds to control the minimum amount of edge energy, minimum luminance of two sequential frames, and noise tolerance. The real-time calculations and the speed of this algorithm are its most significant advantage. However, it is based on a frame-based structure, and it is not easily adapted for event-based cameras.

Cheng and Ridwan [34, 35] designed an optical flow algorithm that takes the input of an event-based camera. Correlations among polarity events are used to identify the object movements. When objects move in a scene, the log-luminance changes mainly occur in the boundary areas in that object. The algorithm can detect motions by comparing the polarities of recent events and previous events. As a result, having a series of pixels with the same polarity in a direction indicates that the object is moving toward that direction. When the object is moving, the pixels of the leading edge will generate the same polarity with the direction of the motion in a period. Therefore, finding events of the same polarity in proximity and within an acceptable period might indicate motion (Figure 3.5).

This algorithm receives the event stream from the event-based camera, and each event contains the timestamp (t), spatial coordinates (x, y), and polarity (p). It generates an optical flow event stream, with each optical flow event containing the timestamp (t), spatial coordinates (x, y), and direction (d) in 8 compass directions.

When this algorithm receives camera events, it searches the eight neighbours of the

location for the recent matching events with the same polarity. This match indicates the direction that the object is moving. Each incoming camera event may match multiple previous events, and one can choose to report the average of the multiple matches, only the first match, or all of the matches depending on the application. Cheng and Ridwan [34, 35] showed that each event can be processed in about 2μs on average, so that it can be done in real-time.

Chapter 4

Motion Compensated Compression algorithm based on Optical Flow

4.1 Introduction

In Chapter 2, two compression algorithms specifically designed for event-based DVS cameras were reviewed. Bi et al. [5, 11] proposed spike coding compression algorithms. However, the spike coding algorithms did not attempt to perform motion compensation. Khan et al. [25] proposed a time aggregation-based lossless video encoding technique for event-based DVS cameras. Though this algorithm performs motion compensation using standard video compression algorithms and obtained better compression ratios, video compression algorithms are not fast enough for real-time applications. Motion compensation in video compression algorithms is slow because it often requires exhaustive search, and it does not take advantage of the event properties in the motion compensation computations.

In this thesis, we want to use faster motion compensation based on event-based optical flow to improve spike coding. We use the event-based optical flow algorithm proposed by Ridwan and Cheng [34, 35] to predict current events from previous events based on motion. In order to improve compression ratios, we do not encode all the events. Instead, the optical flow and unpredicted DVS events are encoded.

In this chapter, we discuss the overall architecture of our proposed lossless Motion Compensated Compression algorithm based on Optical Flow (MCCOF). Spike cube partition and optical flow event prediction procedures will be introduced in later sections. We also describe how context modelling is used with arithmetic coding in our algorithm.

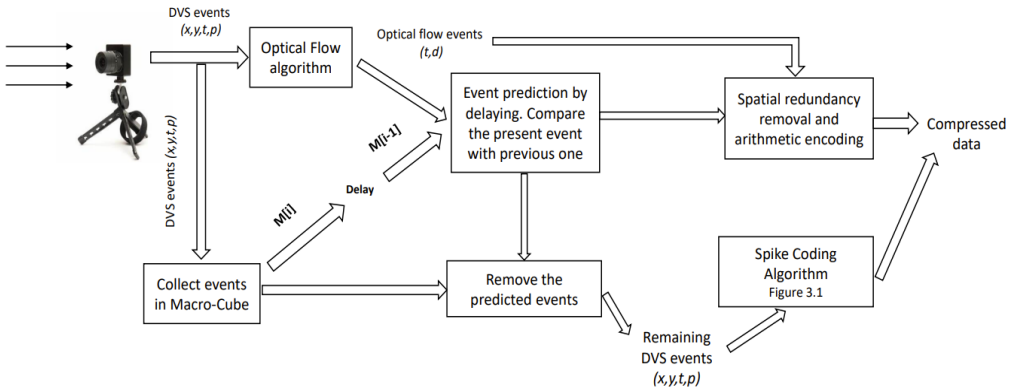

Figure 4.1: Overall architecture of MCCOF.

4.2 Overall Architecture

The polarity events from the DVS camera are processed by the optical flow algorithm to obtain optical flow events. At the same time, the polarity events are also collected into macro-cubes. The optical flow events are used to predict polarity events from previous polarity events. Predicted polarity events are removed, and the remaining polarity events are encoded using the spike coding algorithm. The optical flow events are also encoded using an adaptive context-based arithmetic encoder. The overall proposed architecture of MCCOF is shown in Figure 4.1.

A brief description of the main components of the MCCOF algorithm are given below.

Macro-cube Partitioning The DVS polarity events are collected into fixed size macro-cubes. Each macro-cube is then partitioned spatially into spike-cubes. The spike-cubes are processed one at a time during the encoding process.

Optical Flow Algorithm The DVS polarity events are also processed by the optical flow algorithm [34]. This algorithm generates optical flow events, which are used for polarity event prediction. The optical flow events are also encoded by the arithmetic coder.

Event Prediction In order to properly decode the entire set of DVS polarity events,

both the encoder and decoder need to maintain a data structure representing the reference polarity event and the predicted polarity event for each optical flow event. The proposed prediction structure ensures that the decoder has decoded the reference polarity event before the corresponding optical flow event is used to reconstruct the predicted polarity event.

Spike Coding The predicted polarity events are removed, and the remaining polarity events are encoded by the spike coding algorithm [5].

Arithmetic Encoding The optical flow events and the spike coding outputs will be encoded by the arithmetic encoder. Adaptive context modelling is also used to improve its compression performance.

4.3 Spike-Cube Partitioning

Partitioning events temporally and spatially keeps the events closer together in the macro-cube and spike-cube. It also increases temporal and spatial redundancies, so cube-based coding proposed by Bi et al. [5] and Dong et al. [11] is implemented to remove temporal and spatial redundancies.

Each macro-cube contains a fixed number (M) of polarity events. Expanding this M means fewer macro-cubes, but there may be less temporal redundancies inside each cube. On the other hand, lowering M too much would lead to too few events in macro-cubes to take advantage of temporal redundancies. These macro-cubes have the full spatial resolution of the pixel array.

Each macro-cube is then split spatially into smaller spike-cubes. The parameter S is used to control the number of resulting spike-cubes so that there are $S \times S$ equal spike-cubes. For example, if $S = 4$, the pixel array is split into a grid of 4×4 spike-cubes. This is illustrated in Figure 4.2. Expanding this S may mean more spatial redundancies per spike-cube because the events come closer to each other, but if S is too large, there may be many

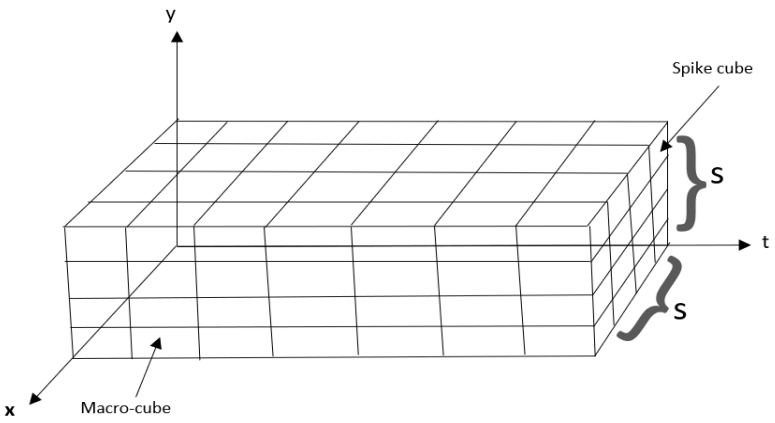

Figure 4.2: Spike-cube partitioning. Events are partitioned into macro-cubes by time, and then partitioned spatially into $S \times S$ spike-cubes.

spike-cubes with very few events.

4.4 Optical Flow Event Prediction

Optical flow is the pattern of apparent motion indicated by a vector of each pixel in a scene. It indicates the motion of a reference polarity event to a predicted polarity event. Although an optical flow event has four components (t, x, y, d), the spatial coordinates (x, y) are the same as those of the reference polarity event and can be omitted. Therefore, it may be easier to encode optical flow events compared to the polarity events, each having two elements (t, d).

When the decoder reconstructs the polarity events from the compressed data, it needs to match each optical flow event with a corresponding reference polarity event that has already been decoded. Specifically, if the encoder always encodes the reference event before the optical flow event, then the decoder can always decode the reference event before decoding the optical flow event. This requirement leads to a data structure that both the encoder and decoder must maintain to decode the polarity events properly. This data structure can be modelled as a directed graph. In this graph, each directed edge represents an optical flow event, and connects the reference polarity event to the predicted polarity event.

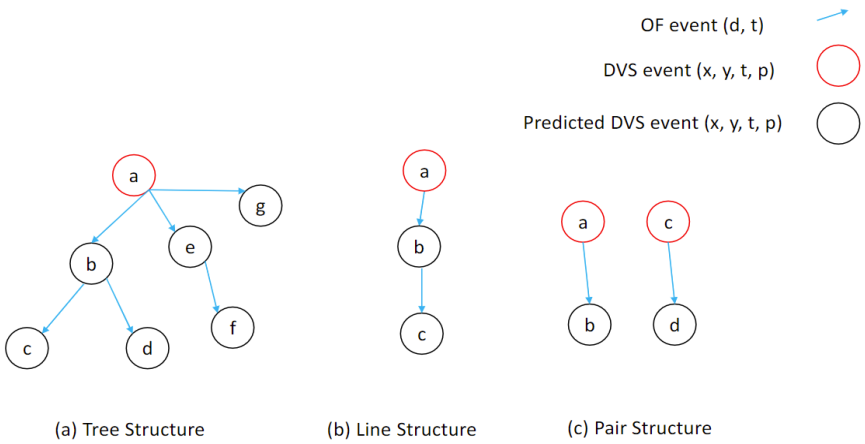

Figure 4.3: Three different prediction graph structures.

The graph structure must also be encoded. We have tried three types of graph structures for our prediction illustrated in Figure 4.3. Their advantages and disadvantages are discussed below.

Tree Structure: Starting from one reference polarity event, optical flow events connect polarity events that are predicted from the reference event directly. These predicted events can in turn be used as reference events to predict other polarity events, and so on. This results in a tree as shown in Figure 4.3(a). In this figure, a is the initial reference polarity event, and with the help of the optical flow event (blue arrow), it predicts event b. Similarly, using b, we can predict events c and d. Similarly, e and g are predicted from a, and f is predicted from e.

There are different ways the tree structure can be encoded. One possible way is to consider how a tree can be traversed. Since we need to process the reference event before the predicted events in both side of the encoder and decoder. Pre-order traversal is chosen to perform the encoding and decoding of the tree. The algorithm is shown in Algorithm 1.

In this algorithm, the events at the roots of the trees are encoded as polarity DVS events from lines 4 to 7. For each subtree, the optical flow event from the root to the subtree

Algorithm 1 Tree Encoding Algorithm

```
1: Input: root: the root of the tree representing the prediction structure. The tree is
   assumed to be non-empty.
   level: current level from the initial reference event
2: Output: last height: depth of the last descendent processed
3: evt ← polarity event at the root
4: if level = 0 then
5:     encode type as POLARITY
6:     encode evt as a polarity event
7: end if
8: last height ← 0
9: for each child of root do
10:    for i = 1, ..., last height do
11:        encode type as MOVE-UP
12:    end for
13:    encode optical flow event from root to child
14:    last height ← the result of recursively encoding the subtree rooted at child with level
       + 1.
15: end for
16: return last height + 1
```

is encoded in line 11, and the subtree is encoded recursively in line 12. In order to move from one subtree to the next subtree, the algorithm needs to encode information to return to the root before descending to the next subtree on line 10. The *last height* returned is used to determine the number of levels to move up to the root. This is used so that we can encode the correct number of steps to move to an ancestor as we traverse the tree.

For example, the root of the tree in Figure 4.3(a) is a and it is encoded as a polarity event. Then the blue arrow is represented as an optical flow event and it is initiated from the tree root a. The node b is the predicted event using optical flow and it is referred to as a child event. This optical flow event is encoded. So the encoding of the tree consists of the polarity event at a, followed by the optical flow event $a \rightarrow b$, $b \rightarrow c$, MOVE-UP, $b \rightarrow d$, MOVE-UP, MOVE-UP, $a \rightarrow e$, $e \rightarrow f$, MOVE-UP, MOVE-UP, $a \rightarrow g$.

The tree prediction structure allows us to predict all events associated with an optical flow event. However, the disadvantage is that it requires the tree's shape to be encoded.

The flexibility of the tree structures may lead to more polarity events being predicted at the expense of a higher tree encoding cost. The trade-off will be examined in our experiments.

Line Structure: It is similar to the tree structure, but each event can only be used to predict one other. The algorithm is shown in Algorithm 2.

Algorithm 2 Line Encoding Process

```
1: start: the start node of the line representing the prediction structure. The line is assumed to be non-empty
   level: current level from the initial reference event
2: evt ← polarity event at the start node.
3: if level = 0 then
4:     encode type as POLARITY
5:     encode evt as a polarity event
6: end if
7: if start has a next node then
8:     encode optical flow event from start to next
9:     recursively encode the remainder of the line starting at next with level + 1
10: end if
```

In this algorithm the events at the starting nodes of the lines are encoded from lines 3 to 6. The optical flow event leading to the remainder of the line is encoded on line 8, and the remainder of the line is recursively encoded on line 9.

For example, the starting node a of the line in Figure 4.3(b) is encoded as a polarity event. Then the blue arrow is represented as an optical flow event and it is initiated from the starting node a. The nodes b and c are the predicted events using optical flow and they are referred as next events. The encoded sequence consists of the polarity event at node a followed by the optical flow events $a \rightarrow b$ and $b \rightarrow c$.

The line prediction structure is easier to encode because each node can predict at most one other. There is no need to move backward in the line the way a tree structure requires. However, the disadvantage is that it limits the number of polarity events that can be predicted. The trade-off will be examined in our experiments.

Pair Structure: Each event is used to predict at most one other or be predicted from another event, but it cannot be both. The algorithm to encode the pair structure is shown in Algorithm 3.

Algorithm 3 Pair Encoding Process

1: **Input:** *start* the reference event of the pair.
2: encode event at *start* as a polarity event
3: encode optical flow event from *start* to predicted event.

In this algorithm the reference event of each pair is encoded as a polarity event, and the predicted event is encoded as an optical flow event.

For example, in Figure 4.3(c), there are two pairs. The nodes a and c are encoded as polarity events. The nodes b and d are the predicted events using optical flow and only the optical flow events are encoded.

The pair prediction structure has the lowest overhead to encode, because the first event is always the polarity event and the second event is always the optical flow event. Therefore, no other encoding is needed to distinguish between reference polarity events and optical flow events. However, the disadvantage is that the number of events predicted is lowest compared to the tree and the line structures. The trade-off will be examined in our experiments.

4.5 Context Modelling for Arithmetic Encoding

We need to use appropriate contexts for entropy coding to encode the symbols more efficiently. Each context is a probability model of the likelihood of each symbol and is used to adapt the data source being encoded. Different contexts are used for different types of symbols encoded. We have used different contexts for timestamps, spatial coordinates, motion direction, polarity, and event types.

4.5.1 Spike Encoding for DVS

For encoding, we need to set different context models for timestamps, spatial coordinates, and polarity so that the models can adapt to the symbol probabilities of each type of data. We used specific types of context models for all those event elements.

The timestamp is a 64-bit quantity from the DVS. For most cases, we encode the difference between the timestamp and that of the previous event. We break the quantity into eight 8-bit blocks—the most significant 8 bits, the next most significant 8 bits, and so on. There are two reasons for partitioning the timestamp into eight 8-bit blocks. First, there are only 256 possible 8-bit symbols so it is possible to record and update the symbol probabilities. If the entire 64-bit timestamp is used, there are 2^{64} different symbols which is impractical to maintain. Moreover, the more significant blocks have different probabilities than the least significant blocks, because the time differences are small. This partitioning allows different context models to adapt to different portions of the timestamp difference.

The spatial coordinates x and y are each encoded using its own context model. The number of different symbols in the context is the number of different possible values for x and y. If a spike-cube is large spatially, then the number of symbols would be larger. In cases where we encode the difference in x and y of an event compared to another one, the same process is used.

The polarity of an event is encoded using two context models. The context model chosen is based on the polarity of the previously encoded polarity event. This improves compression performance because the polarity of an event strongly correlates to the polarity of the previous event [5].

Address-Prior Mode

To encode unpredicted events in Address-Prior mode (Section 3.1), a binary map is first encoded. For each location in the binary map, two possible symbols are used to indicate whether there are events in the location. The surrounding neighbourhood is used to select

a context for each location. The number of non-empty locations in the neighbourhood is used to select the context. Once the binary map is encoded, the non-zero value of the number of events in each location is encoded. This is done using a context for the number of events. Finally, the time differences between consecutive events for a particular location are encoded as described above.

Time-Prior Mode

To encode the unpredicted events in a spike-cube in Time-Prior mode, the spatial coordinates of the centre point are encoded using the method described above. For each polarity event, the difference in spatial coordinates compared to the centre is encoded. This is followed by the encoding the timestamp difference from the previous polarity event, and the polarity.

4.5.2 Optical Flow Encoding

To encode an optical flow event, first the timestamp difference compared to the previous polarity event is encoded. Then the direction of the optical flow event is encoded using a context with 8 possible symbols.

4.5.3 General Control

When the encoder moves from one spike-cube to another, we used a specific context to identify the completion of encoding in one spike-cube. It also helps the decoder to know whether there is more to process or not. We encode all the unpredicted events as polarity events. To improve compression performance, we also use a special symbol to indicate that a spike-cube is empty.

For each spike-cube, the appropriate algorithm in Section 4.4 is used to first encode all events that are part of the chosen prediction structure. The remaining unpredicted events are then encoded using the Address-Prior mode or the Time-Prior mode.

4.6 Adaptive Mode Selection

For each spike-cube, the compression algorithm applies both the Address-Prior mode and the Time-prior mode, and chooses the best one based on compression performance for that cube. As a result, an extra symbol must be encoded for each spike-cube to indicate which mode is used.

We can also choose to combine the two modes with or without optical flow motion compensation:

AP = Yes; TP = No: optical flow prediction is used for AP mode but not for TP mode.

AP = Yes; TP = Yes: optical flow prediction is used for both AP mode and TP mode.

AP = No; TP = Yes: optical flow prediction is used for TP mode but not for AP mode.

AP = No, TP = No: optical flow is not used in any mode.

All the experimental results from these four choices will be presented and analyzed in Chapter 5.

Chapter 5

Experimental Results

5.1 Introduction

To evaluate the performance of our proposed MCCOF algorithm, we have implemented and tested it on one of the datasets other researchers have used. For experimental analysis, we used the PKU-DVS dataset from Peking University, China [31]. Since the spike-coding algorithm [5] was evaluated on the PKU-DVS dataset, we choose to use this dataset so we can directly compare our results against it.

In this chapter, we describe in detail the dataset. We present experimental results on this dataset with varying parameters. We compare the results of the MCCOF algorithm against the spike-coding algorithm, and provide an analysis of our experimental results.

5.2 Dataset Description

For experimental evaluation and comparison, we have obtained the PKU-DVS datasets for spike coding from the National Engineering Laboratory for Video Technology (NELVT), Peking University, China [31].

The PKU-DVS dataset contains a number of event sequences representing the output of the DVS camera in a scene. The scenes are classified into two categories. Class A consists of indoor scenes, while Class B consists of outdoor scenes. These indoor and outdoor scenes have been captured under various circumstances. The scenes are captured both during the day and the night, both near view and distant view, and contains high-speed movements. The number of events in the scenes are shown in Table 5.1 and Table 5.2.

Table 5.1: Indoor scenes with total event number.

Scene	Total Event Number
water-drop	11563244
fluorescent	12839146
lighter	2792140
football	9745102
jump	2375023
game	5918278
pendulum	113683

Table 5.2: Outdoor scenes with total event number.

Scene	Total Event Number
intersection	2016923
pedestrians	25455053
daytime-traffic1	14246080
daytime-traffic2	5525454
night-roadside	17018998
night-traffic	105631857

We have found a mismatch in the total event numbers during our experiments for two outdoor scenes. We have counted 2016923 and 105631857 polarity events for the "intersection" and "night-traffic" scenes, respectively. However, the number of events published by the NELVT were 30483325 and 5423636. We have tried to inquire about this mismatch but failed to get any response at this time. We have used the number of events counted by our software for this thesis. Snapshots of detected optical flow for those scenes are shown in Figure 5.1 and Figure 5.2.

The scenes are described below:

- Waterdrop: this indoor scene represents rain water drops falling on the floor. In this scene, DVS events appear frequently throughout the entire scene and are not localized to any particular location.

- Fluorescent: an indoor scene, we can observe a fluorescent gas-filled tube light. DVS events appear in a specific region close to the light.

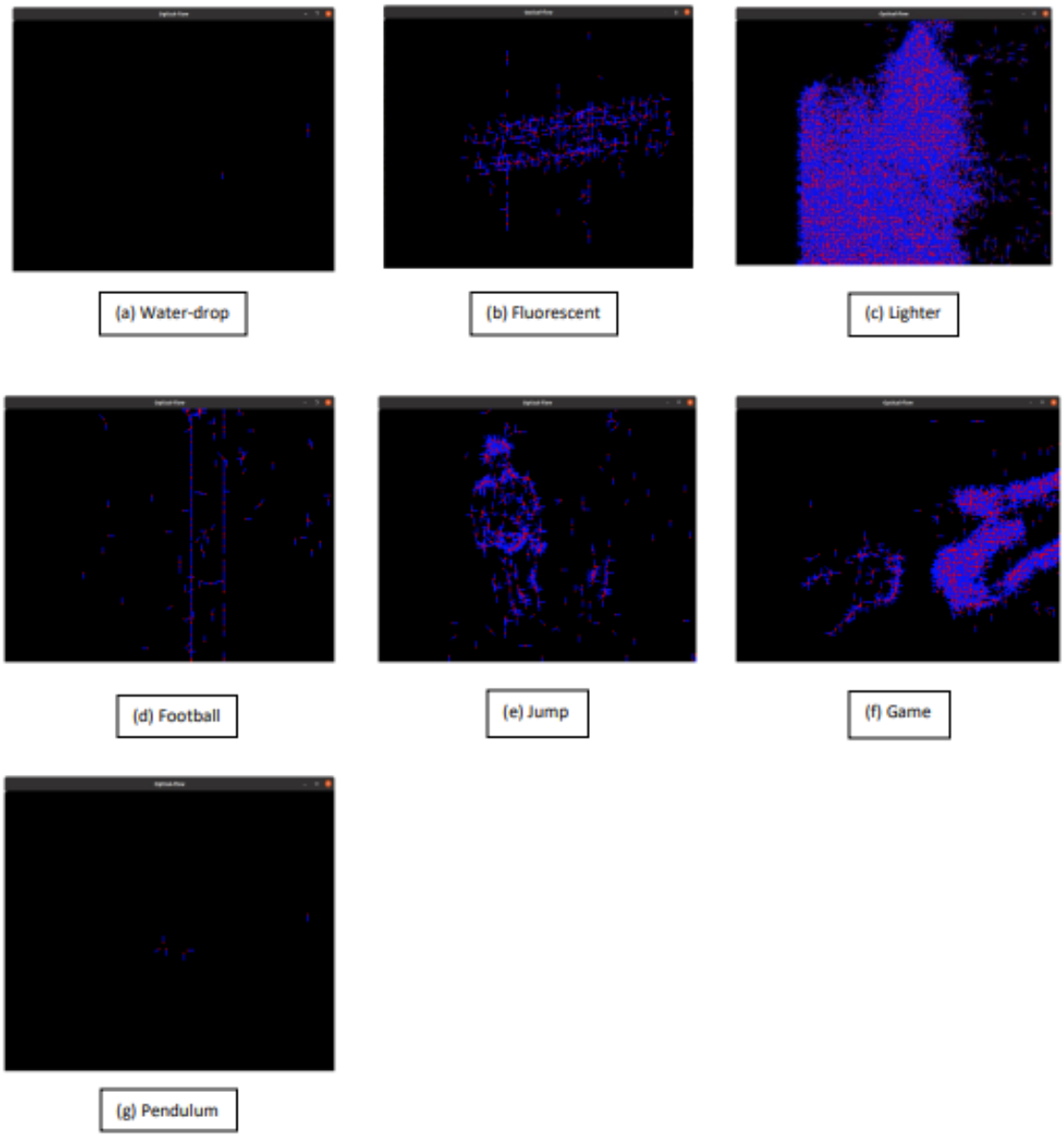

Figure 5.1: Snapshots of detected optical flow for indoor scenes.

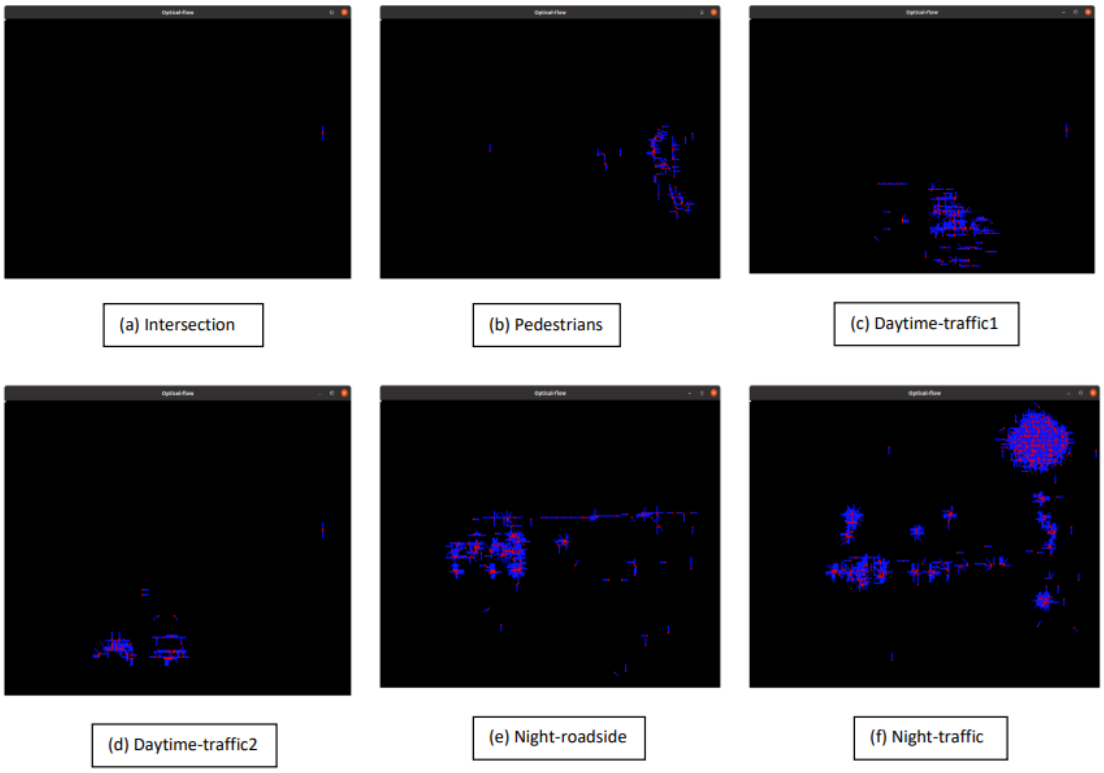

Figure 5.2: Snapshots of detected optical flow for outdoor scenes.

- Lighter: in the indoor Lighter scene, DVS events are generated from the centre area around the flame.
- Football: a bouncing football appears in this indoor scene. DVS events are limited to the area where the ball is bouncing.
- Jump: in this indoor scene, a few people jump in a specific region in the scene. DVS events are concentrated around where the people are.
- Game: in this indoor scene, two people are playing a "rock paper scissors" game. DVS events occur around where the hands are.
- Pendulum: in this indoor scene, a pendulum swings back and forth between the left and right side of the camera view. DVS events are present in the whole scene. However, at any one moment, DVS events only occur around the pendulum.
- Intersection: in this outdoor scene, a single cross is blinking in the scene. DVS events are concentrated around the cross.
- Pedestrians: in this outdoor scene, several pedestrians are crossing a road. Events are spread out all over the region in the scene.
- Daytime-traffic1 and Daytime-traffic2: both outdoor scenes show busy roads with a high amount of traffic. Events are spread out all over the region in the scene.
- Night-roadside: this outdoor scene shows the night view of a roadside full of traffic and pedestrians. Events are spread out all over the region in the scene.
- Night-traffic: this outdoor scene shows the night view of road traffic. It is the largest scene in terms of the number of events among all the other scenes. Events are spread out all over the region in the scene.

Table 5.3: Compression ratios for three prediction structures for indoor scenes.

Scene	Tree	Line	Pair
waterdrop	32.36	50.36	**51.24**
fluorescent	10.74	**26.74**	26.65
lighter	13.60	17.20	**18.01**
football	4.59	15.38	**15.48**
jump	5.40	8.18	**8.34**
game	3.78	4.33	**4.45**
pendulum	3.64	3.89	**4.04**

These scenes have different characteristics, and our experimental results will show that some of these scenes are easier to compress than others. In particular, scenes where the DVS events occur in a spatially clustered way are generally easier to compress.

5.3 Experiments

In this thesis, we implemented the optical flow algorithm [13] for event prediction. We used the PKU-DVS dataset to evaluate our proposed MCCOF algorithm performance. Also, we compared our compression ratio (CR) results with the spike coding algorithm.

5.3.1 Choice of Prediction Structure

To compare the effectiveness of each of the three prediction structures described in Chapter 4, we implemented the tree structure, the line structure, and the pair structure. Since we are only interested in comparing the choice of prediction structures here, we only used TP mode with the same number of events in each macro-cube (40000) and the same number of spike-cubes ($S = 10$).

Compression ratios using these three predictions structures are shown in Table 5.3 and Table 5.4. We observe that for the pair prediction structure, we get higher compression ratios (CR) on all but one scene. The compression ratio for pair prediction on the fluorescent is slightly lower than that for line prediction. However, the difference is very small. In Chapter 4, we discussed the trade-off between reducing the encoding cost of events

Table 5.4: Compression ratios for three prediction structures for outdoor scenes.

Scene	Tree	Line	Pair
intersection	65.07	68.85	**72.63**
pedestrians	11.85	13.66	**13.79**
daytime-traffic1	10.14	11.16	**11.38**
daytime-traffic2	4.26	5.19	**5.34**
night-roadside	4.37	4.83	**5.21**
night-traffic	3.39	3.75	**3.90**

by performing more prediction against the overhead to encode a more flexible prediction structure. The results in Tables 5.3 and 5.4 show that the overhead to encode a more flexible structure such as a tree outweighs the reduction obtained by performing more prediction. The overhead for encoding a tree structure is much higher than the overhead for encoding the line structure or the pair structure, while the difference between the encoding cost of the line structure and the pair structure is lower. While the line prediction structure is simpler, the pair structure requires the least overhead to encode the prediction structure. Based on the results, we decided to focus only on the pair prediction structure for the remainder of the experiments.

5.3.2 Effects of Varying Number of Events in Macro-cubes

Changing the number of events (M) in each macro-cube affects the compression ratio. Increasing M may allow us to take advantage of temporal redundancies better because there are more events in each macro-cube.

To understand the effect of varying the number of events in each macro-cube, we tested our algorithm using $M = 20000$, $M = 40000$, and $M = 65000$. Only pair prediction structure with $S = 10$ is used, and adaptive AP or TP mode is chosen for each spike-cube. The results of varying M for indoor and outdoor scenes are shown in Table 5.5 and Table 5.6. For indoor scenes, using a larger value of M results in higher compression ratios. The reason is that the events are generally clustered, so the larger number of events allow the TP mode to compress the spatial locations of events better.

Table 5.5: Compression ratios for varying the number of events number in each macro-cube for indoor scenes.

Scene	$M = 20000$	$M = 40000$	$M = 65000$
waterdrop	53.769	54.106	**54.235**
fluorescent	31.107	31.836	**32.140**
lighter	21.883	**21.963**	21.960
football	17.893	18.155	**18.289**
jump	9.730	9.807	**9.826**
game	5.247	**5.254**	5.245
pendulum	4.895	4.920	**4.934**

Table 5.6: Compression ratios for varying the number of events number in each macro-cube for outdoor scenes.

Scene	$M = 20000$	$M = 40000$	$M = 65000$
intersection	75.774	76.602	**76.913**
pedestrians	**15.937**	15.870	15.771
Daytime-traffic1	**12.969**	12.957	12.905
Daytime-traffic2	**6.519**	6.472	6.448
Night-roadside	6.284	6.324	**6.345**
Night-traffic	4.640	4.680	**4.710**

On the other hand, a smaller value of M generally results in higher compression ratios for outdoor scenes. The reason is that in outdoor scenes, the events are generally less clustered spatially. Using a smaller value of M results in macro-cubes that are more temporally clustered, and events inside a small time window are more spatially clustered than events that are in a larger time window.

5.3.3 Effects of Varying Number of Spike-cubes

We also want to determine the effect of varying the number of spike-cubes S in each macro-cube. Using a smaller number of spike-cubes results in larger spike-cubes, so that events in each spike-cubes may not be as clustered. On the other hand, a larger value of S may result in too many spike-cubes with very few events, reducing the effectiveness of motion compensation.

We tested our algorithm using $S = 2, 4, 10, 20, 60$. This allows us to test the algorithm from very large spike-cubes to very small spike-cubes. Only pair prediction is used, and M is set to 20000. However, we separately tested AP and TP modes, each with or without optical flow motion compensation. The two modes are tested separately here because the size of the spike-cubes affect whether the events in spike-cubes are spatially clustered. We would expect that when S is large, more spike-cubes would be spatially clustered. The results are shown in Table 5.7 and Table 5.8.

We observe that in AP mode, using larger values of S generally decreases the compression ratio. The decrease is more significant for indoor scenes compared to outdoor scenes. This happens whether motion compensation is used. As S increases, the size of each spike-cube decreases, so that the events are not as spatially decentralized. Therefore the AP mode does not perform as well for larger values of S. For outdoor scenes, the events are not as localized so the effect is not as large.

For TP mode, in most cases, there is a significant increase in compression ratio when S increases. For example, in the lighter scene, the compression ratio increases from 11.24

Table 5.7: Compression ratios for varying the number of spike-cubes for indoor scenes.

Scene	**Mode**	**Motion Compensation**	$S = 2$	$S = 4$	$S = 10$	$S = 20$	$S = 60$
water-drop	AP	yes	**53.21**	53.16	53.04	52.97	52.81
		no	**53.71**	53.66	53.54	53.46	53.45
	TP	yes	50.65	50.29	49.58	49.35	**50.84**
		no	**50.84**	51.21	50.12	49.89	50.82
fluorescent	AP	yes	**26.52**	26.45	26.02	25.69	25.13
		no	**25.59**	25.54	25.30	25.17	24.92
	TP	yes	14.76	**26.65**	25.05	23.51	22.23
		no	15.07	**27.44**	26.64	25.30	23.19
lighter	AP	yes	17.73	18.16	18.54	**18.55**	17.62
		no	17.02	**17.04**	16.98	16.89	16.62
	TP	yes	11.24	17.56	**18.01**	17.70	16.34
		no	11.74	18.94	**20.06**	20.04	18.14
football	AP	yes	**15.31**	15.25	15.07	14.85	14.42
		no	**14.57**	14.55	14.50	14.39	14.21
	TP	yes	14.46	**15.48**	14.67	13.73	12.91
		no	15.02	**16.15**	15.71	14.75	13.52
jump	AP	yes	8.40	**8.42**	**8.42**	8.34	8.07
		no	**8.06**	8.05	8.04	8.00	7.87
	TP	yes	5.94	**8.34**	8.08	7.70	7.21
		no	6.23	**8.78**	8.65	8.30	7.66
game	AP	yes	4.24	4.33	4.44	**4.45**	4.28
		no	**4.16**	**4.16**	4.15	4.13	4.07
	TP	yes	4.03	4.36	**4.45**	4.32	3.93
		no	4.38	4.76	**4.98**	4.87	4.38
pendulum	AP	yes	4.37	4.38	**4.39**	4.37	4.24
		no	**4.66**	**4.66**	**4.66**	4.64	4.58
	TP	yes	3.55	3.94	3.98	3.92	**4.02**
		no	3.79	4.20	4.21	4.16	**4.24**

Table 5.8: Compression ratios for varying the number of spike-cubes for outdoor scenes.

Scene	**Mode**	**Motion Compensation**	$S = 2$	$S = 4$	$S = 10$	$S = 20$	$S = 60$
intersection	AP	yes	74.43	**74.45**	74.44	**74.45**	74.29
		no	**74.68**	74.63	74.58	74.57	74.56
	TP	yes	20.50	69.44	69.69	70.02	**73.25**
		no	20.60	71.40	71.71	72.04	**73.32**
pedestrians	AP	yes	14.12	14.30	14.56	**14.64**	14.11
		no	**14.66**	14.65	14.63	14.61	14.56
	TP	yes	11.45	13.30	13.75	**13.80**	13.67
		no	12.04	13.35	13.84	13.92	**14.56**
daytime-traffic1	AP	yes	11.83	11.95	12.13	**12.18**	11.95
		no	**12.04**	12.03	**12.04**	12.02	11.96
	TP	yes	7.23	10.99	11.33	**11.38**	11.33
		no	7.51	11.43	11.77	11.84	**12.00**
daytime-traffic2	AP	yes	5.66	5.73	5.85	**5.89**	5.57
		no	**6.03**	6.02	6.02	6.02	6.00
	TP	yes	3.74	5.04	5.29	5.33	**5.35**
		no	3.85	4.96	5.16	5.24	**5.74**
night-roadside	AP	yes	5.28	5.33	**5.36**	5.35	5.23
		no	**5.33**	**5.33**	5.31	5.29	5.24
	TP	yes	4.64	4.94	4.97	**5.21**	4.91
		no	5.12	5.42	5.54	**5.82**	5.40
night-traffic	AP	yes	3.77	3.81	3.83	**3.87**	3.74
		no	**3.82**	**3.82**	3.81	3.80	3.77
	TP	yes	3.71	3.84	**3.90**	3.84	3.53
		no	4.21	4.36	4.49	**4.45**	4.05

when $S = 2$ to 16.34 when $S = 60$. Notice that the compression ratio does not always increase when S increases, as the best compression ratio is actually 18.01 when $S = 10$. This is expected because when S increases, each spike-cube is more clustered. This is the scenario that the TP mode is designed for.

We also note that the compression ratios are generally lower when motion compensation is used. This does not necessarily mean that our proposed motion compensation is ineffective in the final algorithm. For these experiments, we have always used AP mode or TP mode for each spike-cube.

5.3.4 Adaptive mode

In this thesis, our proposed algorithm implements the Address Prior (AP) and Time Prior (TP) mode for each spike-cube and chooses the one that gives the best compression ratio for that spike-cube.

We also applied four prediction scenarios to observe which prediction scenario results in a higher compression ratio. We used $M = 20000$ and pair prediction. Since the size of the spike-cubes can affect whether the events are clustered spatially within a spike-cube, the parameter S can affect whether AP mode or TP mode performs better. Therefore, our experiments were performed with different values of S to understand the impact the choice of this parameter has.

Table 5.9 and Table 5.10 show the indoor and outdoor datasets scenarios for choosing the two modes adaptively. Comparing these results to those in Table 5.7 and Table 5.8, we see the advantage of adaptively choosing the two modes for each spike-cube. For example, for the lighter scene, using only AP or TP mode (with or without motion compensation) results in a compression ratio of at most 20.06 when $S = 10$. However, when the adaptive algorithm is used, the compression ratio is 21.88 in the best case when $S = 10$.

For indoor scenes, the best results are often obtained when motion compensation is not used in either mode. However, the football, game, and jump scenes achieve better results

Table 5.9: Compression ratios for the adaptive algorithm for indoor scenes.

Scene	**Motion Compensation**	$S = 2$	$S = 4$	$S = 10$	$S = 20$	$S = 60$
water-drop	AP = yes; TP = no	**53.84**	53.62	53.17	53.02	53.23
	AP = yes; TP = yes	**53.78**	53.54	53.05	52.96	52.82
	AP = no; TP = yes	**54.49**	54.30	53.70	53.52	53.46
	AP = no; TP = no	**54.55**	54.38	53.77	53.56	53.47
fluorescent	AP = yes; TP = no	27.50	29.37	**31.08**	28.95	25.78
	AP = yes; TP = yes	27.33	28.52	**28.76**	26.75	25.27
	AP = no; TP = yes	26.63	27.81	**28.99**	26.88	25.22
	AP = no; TP = no	26.83	28.64	**31.11**	29.03	25.81
lighter	AP = yes; TP = no	19.24	20.08	**21.83**	21.76	19.19
	AP = yes; TP = yes	18.23	18.70	**19.30**	18.93	17.57
	AP= no; TP = yes	17.98	18.51	**19.46**	19.08	17.24
	AP = no; TP = no	18.98	19.81	21.88	**22.16**	19.21
football	AP = yes; TP = no	16.25	**18.13**	18.02	16.52	14.88
	AP = yes; TP = yes	16.04	**17.17**	16.61	15.28	14.48
	AP = no; TP = yes	15.64	**16.73**	16.64	15.35	14.42
	AP = no; TP = no	16.18	17.50	**17.89**	16.51	14.89
jump	AP = yes; TP = no	8.58	9.35	**9.86**	9.21	8.30
	AP = yes; TP = yes	8.50	8.95	**9.08**	8.43	8.07
	AP = no; TP = yes	8.29	8.78	**9.13**	8.47	7.92
	AP = no; TP = no	8.47	9.25	**9.83**	9.21	8.30
game	AP = yes; TP = no	4.66	4.95	**5.25**	5.11	4.54
	AP = yes; TP = yes	4.24	4.54	**4.66**	4.50	4.28
	AP = no; TP = yes	4.32	4.52	**4.66**	4.52	4.13
	AP = no; TP = no	4.63	4.93	**5.25**	5.11	4.55
pendulum	AP = yes; TP = no	4.37	4.34	**4.46**	4.36	4.51
	AP = yes; TP = yes	4.37	**4.40**	4.39	4.33	4.23
	AP = no; TP = yes	4.63	4.83	**4.91**	4.75	4.58
	AP = no; TP = no	4.62	4.84	**4.93**	4.80	4.60

Table 5.10: Compression ratios for the adaptive algorithm for outdoor scenes.

Scene	**Motion Compensation**	$S = 2$	$S = 4$	$S = 10$	$S = 20$	$S = 60$
intersection	AP = yes; TP = no	74.43	**76.93**	75.77	75.77	75.24
	AP = yes; TP = yes	74.43	**75.02**	74.44	74.45	74.33
	AP = no; TP = yes	74.68	**75.40**	74.58	74.57	74.56
	AP = no; TP = no	74.68	**76.93**	75.77	75.77	75.24
pedestrians	AP = yes; TP = no	14.12	14.31	15.30	**15.42**	15.03
	AP = yes; TP = yes	14.12	14.40	14.84	**14.89**	14.08
	AP = no; TP = yes	14.72	14.90	15.50	**15.53**	14.84
	AP = no; TP = no	14.71	14.91	15.94	**15.98**	15.66
daytime-traffic1	AP = yes; TP = no	11.84	12.01	12.57	**12.68**	12.62
	AP = yes; TP = yes	11.84	12.02	**12.25**	12.19	11.95
	AP= no; TP = yes	12.18	12.42	**12.67**	12.65	12.21
	AP = no; TP = no	12.18	12.51	12.97	**12.99**	12.80
daytime-traffic2	AP = yes; TP = no	5.66	5.73	5.85	5.89	**6.04**
	AP = yes; TP = yes	5.66	**5.89**	5.85	**5.89**	5.55
	AP = no; TP = yes	6.09	6.27	**6.35**	**6.35**	6.14
	AP = no; TP = no	6.03	6.33	6.52	**6.60**	6.50
night-roadside	AP = yes; TP = no	5.61	5.90	**6.32**	6.10	5.64
	AP = yes; TP = yes	5.28	5.42	**5.61**	5.45	5.23
	AP = no; TP = yes	5.34	5.52	**5.75**	5.55	5.28
	AP = no; TP = no	5.42	5.89	**6.34**	6.13	5.65
night-traffic	AP = yes; TP = no	4.29	4.51	**4.63**	4.59	4.18
	AP = yes; TP = yes	3.83	3.97	**4.02**	3.94	3.72
	AP = no; TP = yes	3.82	3.98	**4.05**	3.97	3.71
	AP = no; TP = no	4.28	4.51	**4.71**	4.59	4.19

when motion compensation is used for AP mode only. Better results are also obtained with motion compensation for scenes such as fluorescent and lighter when S is small. For outdoor scenes, the best results are often obtained with no motion compensation. However, the results for the intersection scene is the best when motion compensation is used for AP mode only.

For TP mode, we see that it is not worthwhile to perform motion compensation. That is likely due to the fact that if TP mode performs well in a spike-cube, the events are already spatially clustered. Therefore, using the centroid as a prediction is more effective. On the other hand, AP mode works best when events are not spatially clustered, so that motion compensation can sometimes improve performance.

Finally, our experiments in Tables 5.9 and 5.10 adaptively choose TP mode or AP mode for each spike-cube. However, whether motion compensation is used for each mode is fixed for all cubes and is not adaptive.

5.4 Final Algorithm

In the previous sets of experiments, we examined the effect of varying each parameter on the compression ratios. Based on the results from these experiments, we have decided to use $M = 65000$ and $S = 10$ for the final algorithm. Only pair prediction is used. For each spike-cube, we choose one of the four modes (TP/AP, with or without motion compensation) adaptively. Notice that whether motion compensation is used is also adaptive for each cube, unlike the experiment in the previous section.

In Table 5.11 and Table 5.12, we compare the final compression ratios from our MCCOF algorithm against the spike coding algorithm of Bi et al. [5]. The main difference between the two algorithms is that the MCCOF algorithm adaptively chooses for each spike-cube whether motion compensation is used. We see that the MCCOF algorithm performs better in all but three (waterdrop, fluorescent, and football) scenes. In particular, the MCCOF algorithm performs better on all outdoor scenes. Note that, however, the intersection and

Table 5.11: Compression ratios from the spike coding algorithm (Bi et al. [5]) and the MCCOF algorithm for indoor scenes.

Scene	**Total Event Number**	**Spike Coding**	**MCCOF**
waterdrop	11563244	**59.59**	54.74
fluorescent	12839146	**33.67**	32.14
lighter	2792140	20.38	**22.16**
football	9745102	**18.82**	18.44
jump	2375023	8.97	**9.86**
game	5918278	4.50	**5.25**
pendulum	113683	4.44	**4.93**

Table 5.12: Compression ratios from the spike coding algorithm (Bi et al. [5]) and the MCCOF algorithm for outdoor scenes.

Scene	**Total Event Number**	**Spike Coding**	**MCCOF**
intersection	2016923	62.20	**76.93**
pedestrians	25455053	14.25	**15.98**
daytime-traffic1	14246080	11.13	**12.99**
daytime-traffic2	5525454	6.49	**6.57**
night-roadside	17018998	5.44	**6.34**
night-traffic	105631857	3.87	**4.71**

night-traffic scenes contain a different number of events as reported in [5].

From the results of our experiments, we conclude that there is benefit to add motion compensation to the spike coding algorithm of Bi et al. [5]. However, adding motion compensation for every spike-cube does not lead to improved performance. Instead, it is important to only use motion compensation for those spike-cubes that can benefit from it. Therefore, the final adaptive MCCOF algorithm outperforms the other versions of the algorithm we considered.

Chapter 6

Conclusion

In this thesis, we introduced the Motion Compensated Compression algorithm based on Optical Flow (MCCOF). This algorithm is based on the spike coding algorithm of Bi et al. [5] with the addition of motion compensation. Motion compensation is performed using the real-time event-based optical flow algorithm of Ridwan and Cheng [35].

We analyzed the effect of changing each of the parameters in the resulting algorithms, and arrived at a final set of parameters for our MCCOF algorithm. We showed that our algorithm performs better on most scenes in the PKU-DVS dataset compared to the previous algorithm. We showed that motion compensation does not always improve the compression performance. However, when used selectively, motion compensation can result in improvement.

6.1 Contributions

In this thesis, we proposed an improved form of the spike coding technique with motion prediction by Bi et al. [5]. The optical flow algorithm proposed by Ridwan and Cheng [35] is used in this process, and it is the first work to use optical flow to take advantage of motion in compression. Experimental results have shown that our MCCOF algorithm achieves a higher compression ratio in most scenarios, compared to the spike coding algorithm. Moreover, the computational time for compression and decompression is not significantly increased while using an event-based optical flow approach for motion compensation.

6.2 Limitations

We have only tested the MCCOF algorithm against the PKU-DVS dataset, because the previous algorithms were tested using this set. There are not many publicly available datasets for evaluating our algorithm. The results may change using different datasets. Other previously published works were tested against datasets that are not publicly available.

Also, there are disagreements on two of the scenes (intersection and night-traffic) in the PKU-DVS dataset in terms of the number of events that are present. The two scenes in the dataset we obtained contain more events than what has been described in previous works. Unfortunately we are not able to resolve this difference. Therefore, the improvement of our algorithm over the spike-coding algorithm on these two scenes is unclear.

6.3 Future Research Directions

In this section, we outline some possible future research directions to further improve our algorithm.

More testing: our algorithm was only tested on one dataset. Testing on more extensive datasets may give us more insight on the properties of the algorithm and give us a better comparison against other algorithms. Since there are not many publicly available datasets, it may be better to implement the other existing algorithms and evaluate each algorithm on our own datasets.

Lossy compression: we have not examined the possibility of lossy compression in this thesis. It may be useful to consider approximating the event data to provide further reduction in storage requirements.

For example, we may approximate the timestamp or the spatial coordinates of each event, or we may remove some events by subsampling the events. For example, Banerjee et al. [2] partitioned the scene with quadtrees and subsample events based on the

size of each partition. It may be interesting to look at other ways of subsampling events as well as approximating timestamps and spatial coordinates.

Adaptive macro-cube and spike-cube sizes: Dong et al. [11] improved their original algorithm by adaptively choosing the sizes of macro-cubes and spike-cubes. However, as we have shown with our prediction structures, there is a tradeoff between flexibility of the partition and improvement in compression ratio of the data. Examining this tradeoff may lead to further improvement in our algorithm.

Bibliography

[1] J. Alakuijala and Z. Szabadka. Brotli compressed data format. `https://datatracker.ietf.org/doc/html/rfc7932`, March 2022.

[2] S. Banerjee, Z. W. Wang, H. H. Chopp, O. Cossairt, and A. K. Katsaggelos. Lossy event compression based on image-derived quad trees and poisson disk sampling. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 2154–2158, 2021.

[3] P. Bardow, A. Davison, and S. Leutenegger. Simultaneous optical flow and intensity estimation from an event camera. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 884–892, 2016.

[4] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM computing surveys (CSUR)*, 27:433–466, 1995.

[5] Z. Bi, S. Dong, Y. Tian, and T. Huang. Spike coding for dynamic vision sensors. In *2018 Data Compression Conference*, pages 117–126, 2018.

[6] D. Blalock, S. Madden, and J. Guttag. Sprintz: Time series compression for the internet of things. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(3), sep 2018.

[7] K. A. Boahen. A burst-mode word-serial address-event link-i: transmitter design. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(7):1269–1280, 2004.

[8] Y. Collet and E. M. Kucherawy. Zstandard - real-time data compression algorithm. `http://facebook.github.io/zstd/`, March 2022.

[9] I. Daubechies. *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, 1992.

[10] P. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, page 369–378, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[11] S. Dong, Z. Bi, Y. Tian, and T. Huang. Spike coding for dynamic vision sensor in intelligent driving. *IEEE Internet of Things Journal*, 6(1):60–71, 2019.

[12] F. Dramas, S. J. Thorpe, and C. Jouffrais. Artificial vision for the blind: a bio-inspired algorithm for objects and obstacles detection. *International Journal of Image and Graphics*, 10:531–544, 2010.

[13] F. Dramas, S. J. Thorpe, and C. Jouffrais. Artificial vision for the blind: a bio-inspired algorithm for objects and obstacles detection. *International Journal of Image and Graphics*, 10:531–544, 2010.

[14] P. Elias. Predictive coding-i. *IRE Trans. Inf. Theory*, 1:16–24, 1955.

[15] P. Elias. Predictive coding-ii. *IRE Trans. Inf. Theory*, 1:24–33, 1955.

[16] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, 2022.

[17] R. Gonzalez and R. Woods. *Digital Image Processing*. Pearson Prentice Hall, 3rd edition, 2008.

[18] S. H. Gunderson. Snappy: a fast compressor/decompressor. `https://github.com/google/snappy`, March 2022.

[19] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[20] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[21] C. Iaboni, H. Patel, D. Lobo, J. Choi, and P. Abichandani. Event camera based real-time detection and tracking of indoor ground robots. *IEEE Access*, 9:166588–166602, 2021.

[22] inivation. AEDAT file format. `https://gitlab.com/inivation/inivation-docs/blob/master/Software%20user%20guides/AEDAT_file_formats.md#:~:text=AEDAT%20files%20have%20a%20common%2C%20human-readable%20header%20format.,made%20up%20of%20one%20or%20more%20header%20lines.`, February 2022.

[23] inivation. libcaer. `https://gitlab.com/inivation/dv/libcaer`, February 2022.

[24] S. Jayasuriya, O. Gallo, J. Gu, T. Aila, and J. Kautz. Reconstructing intensity images from binary spatial gradient cameras. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 337–343, 2017.

[25] N. Khan, K. Iqbal, and M. G. Martini. Time-aggregation-based lossless video encoding for neuromorphic vision sensor data. *IEEE Internet of Things Journal*, 8(1):596–609, 2021.

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

[27] D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. *Software - Practice and Experience*, 45(1):1–29, 2015.

[28] M. Litzenberger, B. Kohn, A. N. Belbachir, N. Donath, G. Gritsch, H. Garn, C. Posch, and S. Schraml. Estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor. *2006 IEEE Intelligent Transportation Systems Conference*, pages 653–658, 2006.

[29] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, 2003.

[30] M. Martini, N. Khan, Y. Bi, Y. Andreopoulos, H. Saki, and M. Shikh-Bahaei. Challenges and perspectives in neuromorphic-based visual iot systems and networks. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8539–8543, 2020.

[31] National Engineering Laboratory for Video Technology, Peking University. PKU-DVS dataset. `https://www.pkuml.org/resources/pku-dvs.html`, July 2022.

[32] I. Pavlov. Lzma sdk (software development kit). `https://www.7-zip.org/sdk.html`, May 2022.

[33] W. B. Pennebaker and J. L. Mitchell. *JPEG: Still Image Data Compression Standard*. Hardcover – Illustrated, 1st edition, 1992.

[34] I. Ridwan. Looming object detection with event-based cameras. Master's thesis, University of Lethbridge, 2017.

[35] I. Ridwan and H. Cheng. An event-based optical flow algorithm for dynamic vision sensors. In *Image Analysis and Recognition (ICIAR) 2017*, pages 182–189, 2017.

[36] J. Rissanen and G. G. Langdon. Arithmetic coding. *IBM J. Res. Dev.*, 23(2):149–162, mar 1979.

[37] D. Salomon. *Data Compression: The Complete Reference*. Springer, 4th edition, 2007.

[38] S. S. Tsai, D. Chen, G. Takacs, V. Chandrasekhar, M. Makar, R. Grzeszczuk, and B. Girod. Improved coding for image feature location information. In Andrew G. Tescher, editor, *Applications of Digital Image Processing XXXV*, volume 8499, pages 499 – 509. International Society for Optics and Photonics, SPIE, 2012.

[39] D. Wu, Y.T. Hou, W. Zhu, Ya. Zhang, and J.M. Peha. Streaming video over the internet: approaches and directions. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):282–300, 2001.

[40] J. Yeh, M. Vetterli, and M. Khansari. Motion compensation of motion vectors. In *Proceedings., International Conference on Image Processing*, volume 1, pages 574–577 vol.1, 1995.