POLITECNICO DI TORINO

Master Degree in Communications and Computer Networks Engineering

Master Thesis

Event-based camera communications: a measurement-based analysis

Advisor:
Prof. Paolo Giaccone

Candidate:
Giulia Attanasio

External Supervisor
Dr. Claudio Fiandrino
Dr. Joerg Widmer

July 2019

To M.
and to my Parents.

Summary

Address Event Sensing (AES) represents a completely new way of sensing reality. Legacy vision systems rely on sensors that capture intensity images at a constant rate (e.g., 30 fps) leading to high latency and redundancy in terms of information and computation. This results in wasting precious resources such as energy, CPU, memory access, storage and waiting time for superfluous frames. AES silicon retina instead are bio-inspired sensors that asynchronously emit spikes, i.e., events, whenever they occur. An ON/OFF event is a positive/negative log intensity change that is registered at a pixel whenever a brightness threshold is crossed. Since the concept of frame is completely eliminated, this approach allows to significantly reduce the latency between photodiode illumination change detection and its off-chip transmission down to ≈ 15 μs. By contrast, the time it takes to build a single frame is 33.3 ms for a video at 30 fps resolution. The impact of temporal redundancy removal is such that the low computational requirements make these dynamic vision sensors (also known as event cameras) well suited for real-time feedback vision-based robotics systems. Address-event silicon retina apply to important and trending computer vision and vision-based robotic systems tasks such as multi-object tracking, gesture recognition, autonomous driving and surveillance just to name but a few. To this date, the vast majority of applications of AES are bounded to local devices. For example, the Istituto Italiano di Tecnologia (IIT)'s iCub humanoid robot features two event cameras as eyes to track ball movements. In this thesis, we aim at going beyond and study the case when these events have to be transmitted to remote devices. The potential of such paradigm change is huge and stems from industrial applications to coordination of drone swarms. Specifically, in this work we assess whether

today wireless networks can support event-based communications for specific computer vision tasks such as continuous object tracking. We pay particular attention to the challenges introduced by latency and losses for a distributed event-based video streaming system and compared with legacy vision frame-based systems both over 802.11ac. For the latter, a custom Java platform has been developed to perform the frame-based streaming analysis and object tracking through OpenCV . For the event-based analysis instead, we relied on jAER as rendering support. jAER is an open-source Java-based framework suitable not only for the visualization of real-time events collected through a sensor or recorded datasets, but also for the development of real-time event-based algorithms and applications. The results highlight that a too small buffer size is detrimental for latency and tracking performance because more than the 50% of events are dropped as the application threads involved are not able to keep up with events generation and packetization. The scene complexity and speed of objects are key features for event generation. Events are generated asynchronously and independently one from each other, but at application level, object tracking is performed over cluster of events because these are correlated and define the same moving object. We show that complex scenes with high speed moving objects generates more events and thus are more sensible to losses. If a significant loss of events is experienced within a cluster, the accuracy performance of a continuous object tracking algorithm is affected negatively.

Acknowledgements

Sebbene questa sia stata l'ultima pagina scritta, credo che fosse stata già scritta da tempo. Ci sono innumerevoli persone che hanno contribuito alla realizzazione di quest' opera. Ringrazio il mio relatore Paolo Giaccone, per la disponibilità e per l'opportunità ricevuta e il mio supervisor Claudio Fiandrino per il sostegno e la curiosità trasmessa per la ricerca. Curiosità che poi è diventata il mio lavoro. Vorrei ringraziare i miei colleghi di Imdea, senza i quali l'ambiente lavorativo non sarebbe lo stesso e i miei colleghi del Politecnico, compagni di avventure e sventure, in particolare Angelo per la pazienza, i consigli e la guida fra qualche Hard Disk caduto e qualche pezzo da museo a 32 bit. Vorrei ringraziare i miei amici che per 5 anni hanno atteso periodicamente con ansia il mio ritorno dalla terra di nessuno, i.e., sessione esami, dopo mesi di assenza. Grazie ad Umberto e ad M., da sempre, il processo di santificazione è già stato avviato. Il ringraziamento finale va alla mia famiglia. Ai miei fratelli che hanno lasciato orme affinchè potessi seguirle, e ai miei genitori che hanno creduto in me prima ancora che potessi farlo io.

Contents

Chapter 1

Introduction

1.1 Motivation

By 2021 more than 99 million AR/VR devices will be shipped and the market will reach more than 108 billion dollar [87, 82]. The growing interest in the Augmented Reality (AR) and Mixed Reality (MR) is encouraging progress in the field, leading the sector to a significant growth, it is thus possible to state that this new way of perceiving and interacting with reality will become an essential part of people's daily life. In particular in MR systems merging real and virtual world requires a deep and fast understanding of the surrounding entities in order to allow the final user to interact with both real and virtual objects. Furthermore, in the Industry 4.0 revolution [4, 44, 32, 65] , companies automation relies on AR not only in the design or commissioning, but also in the manufacturing and quality control, embracing every step of the industrial process with the main advantage of reducing the number of errors. Autonomous Driving represents an important scenario in which AR finds application that allows to highlight the importance of latency requirements. An Autonomous Driving system must be able to detect the surroundings vehicles and the vulnerable users almost in real time in order to guarantee a safe driving experience. For example in [49], Deep Neural Networks have been exploited together with Events camera for motion-estimation in order to predict the vehicle's steering angle. Since the increasing number of sensors is affecting different fields, there is an unprecedented enormous amount of video and images generated in real time that must be processed as fast as possible in order to perform some specific tasks, e.g., image tracking or object recognition. When we talk about latency requirements, we refer at first to the interval of time that elapses between the motion performed by the user and the proper content to be displayed by the MR system, commonly known as motion-to-phantom latency that according to experiments has been fixed to 20 ms in order to avoid user sickness. Furthermore we have to consider the time required to perform the specific Computer Vision (CV) task and the rendering time required to generate the image. High latency may affect accuracy of object detection/recognition in the sense that for a rapidly evolving scenario the discrepancy between the reality and the achieved object tracking must be brought to the minimum in order to guarantee that the object will be actually tracked in the correct position. If we consider a feedback-system, e.g., a remote user sending a feedback command according to data received by the sensors, the scenario becomes even more complex due to the fact that also the time needed to trigger back this command must be taken into account. Summarizing we are dealing with an enormous amount of data generated in real time that involves heavy computations and we need to match at the same time low latency constraints, high accuracy and low processor load. According to these system requirements, offloading major computer vision tasks to the edge datacenter represents an interesting possibility especially for mobile devices with limited computation capacity. The edge with respect to the cloud platform introduces a gain in terms of response latency and bandwidth constraints [94]. Address-event camera are bio-inspired sensors that asynchronously emit events, whenever they occur. The impact of their frame-free nature which involves a temporal redundancy removal is so that the low computational requirements and the fast response time make event-camera sensors well suited to map reality evolution into data on which real time edge computing CV tasks which involve AR/MR can be applied.

1.2 Contribution

This thesis investigates the feasibility of a novel solution for the edge processing of real time CV tasks which exploit data collected and directly streamed by an event-based camera. Even if this kind of neuromorphic sensors is available and accessible to most research groups, event-driven computer vision algorithms have not been investigated as deep as traditional frame-based ones, thus the lack in the field is consistent. Furthermore we have to remark that the concept of frame has been completely eliminated, we are streaming events in real time, i.e., the coordinate of an ON/OFF brightness change registered at a pixel. In this scenario traditional frame-based video streaming protocols such as ABR or DASH cannot been exploited, which lead this task to be considered as an open research problem. The main novelty introduced by this work is the fact that for the first time in literature one could consider as an option the streaming of events over the network, to this date, event-based camera have always been employed on board or in close proximity of the device performing actuation. Since this approach has never been explored before, it is fundamental at first to evaluate network performance. By exploiting UDP over a Java-based framework, called jAER, I have performed a series of measurements at first in a two hosts wired LAN then moving to 802.11ac . This preliminary analysis is fundamental in order to understand how events are actually streamed over the network, in this way one can determine which are the network characteristics that must be guaranteed in order to successfully meet latency and accuracy. The metrics have been evaluated by analyzing different datasets that have been recorded with a real event-based camera. The considered scenarios differ for motion speed, acceleration and surrounding in order to highlight the fact that events are generated asynchronously with higher or lower rate according to the motion of the surrounding reality. Event-based object tracking represents a difficult task, since events alone may be ambiguous in the image plane. Thus, different information must be taken into account, [42] such as the cluster size,i.e., the number of events registered within a cluster, or the event lifetime, as proposed by [57], which measures the time required by the moving edge that generate the event to traverse a distance of 1 pixel. In this way, it is possible to encode the information on the amount of time required to continuously represent an event. Furthermore background noise in an heavy textured environment makes object detection a difficult challenge. Different experiments have been performed under different scenarios in order to understand how the network behaves with respect to the streaming of events. In this way, a failure point for the achievement of a specific CV task can be found. By considering the results, it is possible to define a standard approach for the design of a streaming system that exploits edge computing capabilities.

1.3 Organization

The thesis is organized as follows:

- Chapter 2 describes how event-based camera operate and highlights the differences with respect to legacy vision sensors. It introduces the edge computing concept by considering the offloading of CV tasks to the edge.
- Chapter 3 illustrates Java-based jAER framework set-up and experiments implementation. Then it consider a preliminary local analysis of different event datasets. It illustrates the RTP frame-based platform implementation. Finally it provides a description of the benchmark analysis performed through VisualVM Java profiler.
- Chapter 4 analyzes the streaming of events both in wireless and non wireless scenarios, then it presents network-related results detailing the impact of events/packets losses and scene complexity.
- Chapter 5 presents the results in terms of throughput and CPU usage with the ultimate objective of providing a comparison between event-based and frame-based systems. It also consider object tracking in both scenarios.
- Chapter 6 outlines future research directions on the topic.
- Chapter 7 concludes the work.

Chapter 2

Background and Related Work

2.1 Computer Vision

Realtime Computer Vision (CV) represents an emerging field suitable for a wide variety of applications ranging from medicine and biometrics to automotive and augmented reality. The main idea behind this growing field is to automate all the tasks that can be performed by the human visual system. Relevant information are extracted from images and a deep meaning is assigned to them. Compared to simple images obtained through common camera sensors, the high-level understanding of the source scene achieved by CV provides additional information that can be used for example to send feedback and thus control a robotic system. In Robot Vision, both a camera sensor and a CV algorithm allows to fulfill a robotic task such as lifting an heavy weight, sorting and inspecting of work pieces. A realtime scenario requires to complete the processing of a single frame within $30 - 40$ ms and even simple algorithms, e.g., face or smile detection, require a lot of power resulting particularly challenging for mobile and embedded devices [69]. The increasing interest in the field will for sure leads computer vision-based system to become as ubiquitous as touch interfaces. Strict timings and latency vision-based robotic systems need in general an high frame rate and a considerable computational cost in order to meet deadlines and operate almost in real-time [14, 10]. For a traditional camera sensor, the high frame would produce an enormous amount of data so that only few instructions per pixel would be available at this point, exacerbating complexity and CPU load. In some particular scenarios, which for example involve drones or autonomous driving, it is important to exactly and continuously keep track of the environment in order to estimate the position of a moving object [20, 21]. In this case, if a frame-based legacy vision sensor is used, then a fast moving object within a frame will produce motion blur that can significantly degrade quality, thus performance. In Event-based vision sensors instead, the concept of frame is completely eliminated, the temporal redundancy is removed and computational requirements are lowered. All these advantages make address-event sensors well suited for real-time feedback vision-based robotics systems. Event Camera are supposed to fulfill computer vision tasks in highly challenging cases which are inaccessible to legacy vision sensors that involve high speed and high dynamic range scenarios. In standard vision sensors, reality is captured as a sequence of snapshots leading to redundant information, computation and high latency. Event-based camera solve the problem by exploiting a technology that works like the human retina since only pixel changes are transmitted at the time they occur. Table 2.1 highlights the main differences between an event-based camera and a legacy vision sensor [91, 74].

Table 2.1: frame-based camera and event-camera comparison.

LEGACY SENSOR	EVENT CAMERA
Frames	Events
Redundant data	Redundancy free
Resource intensive	Lightweight
High power consumption	Low power consumption
Motion blur	No motion blur
low dynamic range	high dynamic range
high latency	low latency

2.1.1 Legacy Vision Sensors

As Lichtsteiner et al. state in [45], in legacy vision sensor a series of snapshots are taken at a constant rate in order to capture reality leading to high latency and redundancy in terms of information and computation, wasting precious resources such as energy, CPU, memory access, storage and time waiting for superfluous frames. In fact, in a frame by frame working system, complex and heavy processing operation must be applied to the received inputs. Robotic vision-based tasks which have strict timing requirements, cannot successfully exploit traditional video frame-based camera because of the heavy amount of data generated and the high frame rate requirements. The human eye can process and perceive individually from 10 to 12 images per second. If we increase the frequency at which the frames are captured then the results is that of continuous motion perception. In general uniform modulated light is perceived as stable from 50 to 90 Hz while hundreds of Hz would be necessary in the case of unmodulated light [12]. High performance cameras found applications in different fields, ranging from detection and tracking from air based platform, to high performance scanning required by industry. According to their specifications, these high performance camera will emit frame at a specific constant rate, see Figure 2.1. In some particular scenarios in which latency represents a problem, e.g.robot vision algorithms, this leads to :

1. *Redundancy in terms of useful information* : A frame is by far bigger with respect to an event, furthermore for slow evolving scenario or stable background reality, subsequent frames are quite similar. Thus, either frame redundancy is filtered out or precious resources in terms of storage and memory access are wasted.

2. *Redundancy in terms of computation* : As stated previously, subsequent frames may be very similar. If redundancy is not filtered out, then the same processing must be applied over the same data.

3. *High Latency* governed by the frame rate no matter the input data. (see 2.1.2 for further clarifications on event representation).

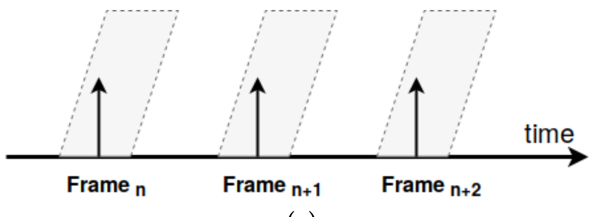


Figure 2.1: (a) reality is captured at constant framerate by a standard camera (b) sequence of frames of a video.

Preliminaries on OpenCV and object tracking

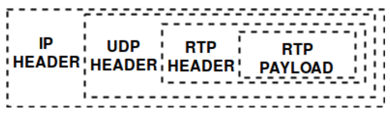
Open Source Computer Vision, formally OpenCV, is a crossplatform library of programming functions mainly aimed at real-time computer vision [69]. Gary Bradsky started the project at Intel in 1999, and came up with the first release in 2000. It has a modular architecture thus it is composed by several modules [63]:

- *Core*: defines the basic data structures such as the Mat multi-dimensional array and some other functions used by other modules.
- *Imgproc*: this module for image processing allows to perform linear and not linear filtering and some other transformations like resize, histogram evaluation and conversion between color space.
- *Video*: this is the module allows video processing defining object tracking algorithms and motion estimation.
- *Calib3d*: this module allows to perform stereo camera calibration, object pose estimation and 3D reconstruction.
- *Features2d* : includes feature detectors and descriptors that allow the detection of some objects belonging to some well defined classes like cars, faces, balls and so on.

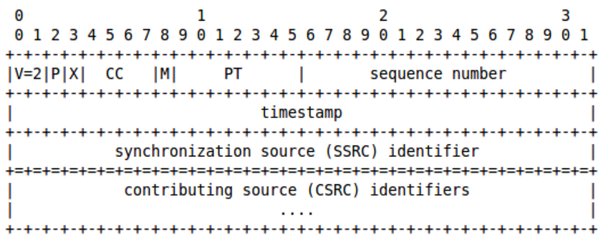
Tracking represents an opportunity for image recognition too. In general a tracking algorithm exploits the knowledge of direction and speed motion of an object according to detection performed in the previous frames. In this way a tracking algorithm is overall faster with respect to a detection algorithm. Whenever partial occlusion of a moving object occurs, an object recognition algorithm will probably fail, instead a good tracker can tolerate occlusion up to a given threshold, avoiding performance degradation. In most instances a tracking algorithm exploits the information contained only in the previous frame while an object recognition one starts from scratch every time it is performed. Thus we can combine the strengths of both approaches. Object tracking by the way, accumulate errors so that the variance of the rectangular cluster estimator increases and the accuracy of the continuous tracking decreases. In order to overcome the problem a detection algorithm is occasionally run. While OpenCV tracking API was introduced in OpenCV 3.0. 8 different trackers are available in OpenCV 3.4.1 : BOOSTING, MIL, KCF, TLD, MEDIANFLOW, GOTURN, MOSSE and CSRT, [5, 43]. In general when one has to perform object tracking, he needs to estimate the position of an object within a frame, by supposing to have correctly tracked it in the previous ones. Whenever object tracking is performed, the motion model, i.e., direction of motion and speed of the object, is derived and can be used in order to determine the position of the target in the next frames. Furthermore, since the object is characterized by some features, an appearance model can be derived too. This model can be used in order to more precisely predict the position of the target. Most of the times in real life the appearance of an object represents a not static component since it can suddenly change. To overcome this problem online training of the classifier is performed. In general a score between 1 and 0 is returned by a classifier, this score represents the probability that the rectangular patch of an image contains or not the object. The classifier will output 0 if it is sure that the patch does not correspond to the object while it is referred to the background and 1 in the opposite case.

RTP : Realtime Transport Protocol notions

An RTP Client and Server Java-based platform has beed adjusted in order to perform frame-based streaming analysis and tracking, see Section 3.3 for the results. The Real-Time Transport Protocol (RTP) is an Internet protocol which provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video [6]. Those services include payload type identification, sequence numbering, timestamping and delivery monitoring. Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services; both protocols are contributing parts of the transport protocol functionality. The protocol specifies a way for programs to manage the real-time transmission of multimedia data over either unicast or multicast network services [75, 37]. It is located at the transport layer of the OSI model together with UDP, Figure 2.2, but from a developer's point of view it belongs to the application layer since the programmer should implement it at the application level in order to allow the encapsulation of RTP packet within UDP. RTP is tipically used in conjunction with RTCP that allows the synchronization of multiple stream and to monitor Quality of Service and transmission statistics. In general realtime multimedia streaming application can tolerate losses in order to have realtime guarantees [59]. Within this scenario losses can causes occasional glitches in the audio or video playback that can be somehow recovered through some error concealment algorithms [71]. TCP has been standardized for RTP but in general is not used since it aims at providing reliability which negatively affects applications with timing constraints because of time-consuming retransmission process. The information contained in the RTP header, see Figure 2.2, allows the receiver to reconstruct the data and describe the encoded format of the data. A signalling protocol like SIP, Jingle, H.323 or RTSP it is typically used to initiate a RTP session. In general two different RTP sessions are initiated for audio and video so that a receiver can opportunely decide which stream to receive, thus tailoring the stream to the bandwidth of the recipients.



(a)



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(b)

Figure 2.2: (a) RTP payload encapsulation (b) RTP header.

The RTP header, 2.2, accounts for [75]:

- *version (V)*: 2 bits that allow to identify the version of RTP used.
- *padding (P)*: this padding bit if set, indicates that the packet contains one or more additional padding octets at the end which are not part of the payload.
- *extension (X)*: 1 bit that indicates whenever the header must be followed by some header extension according to the list available in RFC 3550.
- *CSRC count (CC)*: 4 bits that indicate the number of contributing source (CSRC) identifiers that follows the fixed header.
- *marker (M)*: the interpretation of the marker over 1 bit is defined by a profile.
- *payload type (PT)*: the 7 bits of this field identify the format of the RTP payload and determine its interpretation by the application, e.g., 26 stays for Motion JPEG, 31 for H.261, the complete list is available at [72].
- *sequence number*: the SN over 16 bits is incremented by one for each RTP data packet sent, and may be used by the receiver to detect losses.
- *timestamp*: the 32 bits of TS indicate the sampling instant of the first octet in the RTP data packet.
- *SSRC*: the 32 bits of the random synchronization source identifier.
- *CSRC*: the 32 bits that identify the contributing sources for the payload contained in this packet.

MPEG-4 part 14

MPEG-4 part 14 or MP4 is a digital multimedia container format used in general to store video and audio but also subtitles and static images. This format is used also for streaming video over the internet. The MP4 compression is lossless, thus audio and video quality of the actual data is not decreased. Its formal designation is ISO/IEC 14496 and was finalized in October 1998 and became an international standard the year after [55]. The aim behind the MP4 development was that of defining an audio and video format that allows to consume low bandwidth. It is a multimedia container thus it does not have a standard method for coding video or audio data. The video files using MP4 are compressed using H.264 or MPEG-4 compression techniques [64]. MP4 provides less loss of quality and higher degree of compression, it can be simply decomposed into constituent objects and it support a wide range of hardware. The popularity gained by MP4s was mainly due to the Apple's iPod. Users at the beginning could download video from Google Video to their personal iPods by using MP4, furthermore Sony PSP made MP4 a popular video format too. MP4 is more popular than Apple's Quicktime format (.mov) because it uses less space and can be played using shareware or freeware softwares such as VLC [54]. As source video for the RTP Client and Server streaming platform, Section 3.3, an MP4 video as been obtained through `ffmpeg` [18], starting from jpg images. The first source video has been obtained from the the VOT2015 Dataset [90] while the second from the jpeg images of the event-camera dataset provided by the ETH of Zurich [36], see Figure 2.3 and Figure 2.4.

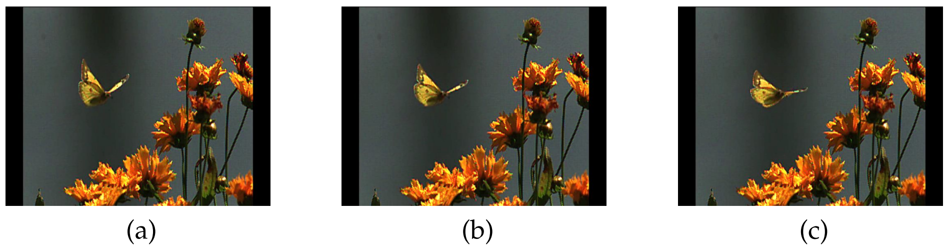


Figure 2.3: three subsequent .jpeg frames of the butterfly dataset of VOT2015.

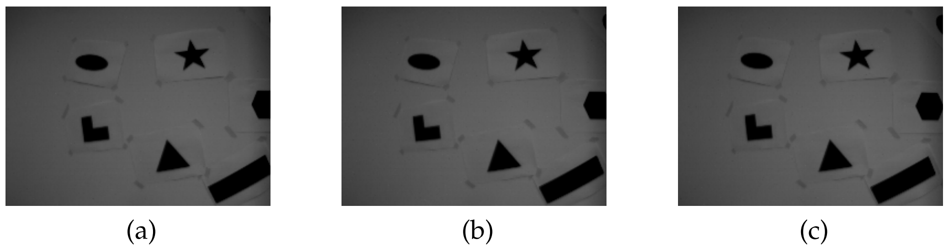


Figure 2.4: three subsequent .png frames of the shapes dataset of the ETH of Zurich.

In particular the butterfly sequence, Figure 2.3, contains 151 .jpeg frames, collected at 30 fps, of size 640×480 pixels. In this simple scenario the background is uniform and static and the main motion is due to the flying butterfly and to flowers moved by the wind. For the sake of simplicity, this scenario has been considered in order to perform object tracking. The shapes rotation sequence, Figure 2.4, contains instead 1356 .png frames of size 240 × 180 pixels. This dataset has been registered at increasing speed by moving a camera in front six simple shapes represented on a wall.

2.1.2 Event Cameras

Address-event silicon retina are bio-inspired sensors that asynchronously emit spikes, i.e., events, whenever they occur, Figure 2.5. An ON/OFF event is a positive/negative log intensity change that is registered at a pixel whenever a certain threshold is crossed. An event encodes time, location and sign of the brightness change. The threshold or more specifically the contrast sensitivity is generally set to the 10 − 15 % and it is possible to relax it in order to limit the event rate as needed. For lower values of the contrast threshold, less events will be generated, for higher values of the contrast threshold instead, the number of generated events increases. As stated in [28], an event-based camera is made by asynchronous pixels that are sensitive to changes in their logarithmic photocurrent $L \doteq log(I)$. An event e_k is generated at time t_k, whenever the brightness change registered at a pixel crosses the contrast threshold $\pm C$ with respect to the previous registered event:

$$\Delta L(x_k, t_k) \doteq L(x_k, t_k) - L(x_k, t_k - \Delta t_k) = p_k C, \tag{2.1}$$

where δt_k represents the time elapsed since last captured event at the same pixel position, and $p_k \in \{-1, 1\}$ represents the brightness change polarity. If the time interval between two consecutive events at the same pixel location is small, we can approximate the brightness change according to Taylor expansion as follows:

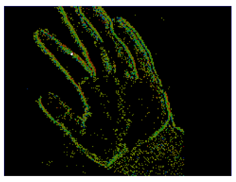
$$\Delta L(x_k, t_k) \approx \frac{\delta L}{\delta t}(x_k, t_k)\Delta t_k. \tag{2.2}$$

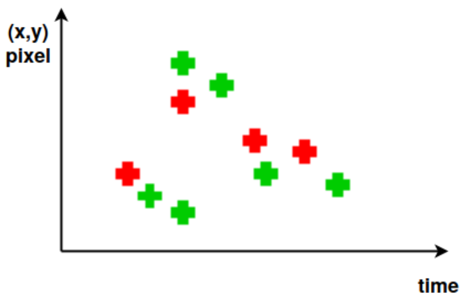
According to previous considerations, events provide information about the temporal derivative of brightness:

$$\frac{\delta L}{\delta t}(x_k, t_k) \approx \frac{p_k C}{\Delta t_k}. \tag{2.3}$$

Temporal redundancy is naturally filtered out since events are generated by moving edges only, furthermore the delay is significantly reduced since it is not necessary to wait for the entire frame to be processed. The advantage is particularly significant whenever one deals with static scenarios. In these cases most of the time the pixel values are unchanged but still must be processed. It is furthermore important to remark that asynchrony means that it is possible to work at a pixel response time precision which is obviously faster with respect to the traditional quantization frame rate. The frame-based to event-base revolution guarantees extraordinary advantages of whom different sectors would not do without, it embraces important and trending computer vision and vision-based robotic systems tasks such as multi-object tracking, gesture recognition, autonomous driving and surveillance just to name but a few, as showed by the Robotic Goalie developed in [14] and the pole balancing arm proposed in [10].

(a)


(b)



(c)

Figure 2.5: (a) DVS128 (b) events rendering through jAER (c) asyncronous ON(red)/ OFF(green) events generation.

Event Representation

Referring to [45], in Figure 2.6(a) the chip format of a 128 × 128 pixel array 4M2P DVS is presented. A pixel produces address events whenever it registers a log intensity change greater than a certain threshold T since the last registered event.

The event at a pixel position will be transmitted as an address in the form of 7 bits for the x row, 7 bits for the y column plus 1 bit for the ON/OFF polarity change, for a total of 15 bits. To the 15 bits describing the event, are added a synchronization bit, rarely used, and a timestamp of 14 bits computed by a counter within the USB-AE converter, for a total of 4 bytes. Figure 2.7 of [24], clarifies the common events structure and the DVS to Host system set up exploited for computer vision based robotic tasks [14, 10]. The pixels are subjected to an adjustable refractory period which prevent them from generating another event within this interval. In this way a small group of pixels will not be allowed to take the entire full capacity. Then events are queued to be transmitted off-chip and losslessly serviced by an arbiter.

We can state that for a smoothly variation of the temporal contrast the event rate can be expressed as follows [45] :

$$Event\ Rate(t) \approx \frac{TCON(t)}{\theta} = \frac{1}{\theta}\frac{d}{dt}ln(I), \tag{2.4}$$

where θ is the temporal contrast threshold that depends on the settings of the comparator and TCON(t) is the temporal contrast which depends on the photocurrent as:

$$TCON(t) = \frac{1}{I(t)}\frac{dI(t)}{dt} = \frac{d(ln(I(t)))}{dt}. \tag{2.5}$$

The surrounding reality is obviously the source that feeds the sensor, more or less faster according to the considered scenario. This means that a scene taken by a camera mounted on a flying drone, for example, will produce more events / s, with respect to a static scene in which only few brightness changes are recorded. For instance one can think at a surveillance event camera mounted at a fixed point, in which the only brightness change is given by the sporadic wind causing the movements of leafs. If we introduce a T_{frame} time parameter referring to frame-based system and an analogue $T_{reality}$ [67], we can distinguish between two peculiar scenarios:

1. $T_{reality} > T_{frame}$: reality is slow evolving compared to frame rate, thus heavy processing is repeated over quite similar/redundant inputs.

2. $T_{reality} < T_{frame}$: reality is fast evolving compared to frame rate, motion blur might be managed and furthermore it might be difficult to perform some image processing tasks.

In an ideal system, the perfect solution would be that of adjusting the time parameters so that $T_{reality} \sim T_{frame}$, in this way redundancy and computing resources can be properly managed. Instead in event-based systems, pixels produce spikes with microsecond delay according to intensity changes, we can thus talk about pseudo-simultaneity referring to the fact that the time parameters naturally match $T_{reality}$. The AE sensor will translate the brightness change into an ON/OFF event as previously explained. In Figure 2.6(b) from [58], the output of a rotating disc stimulus produced by both a standard camera and a Dynamic Vision Sensor, is presented.

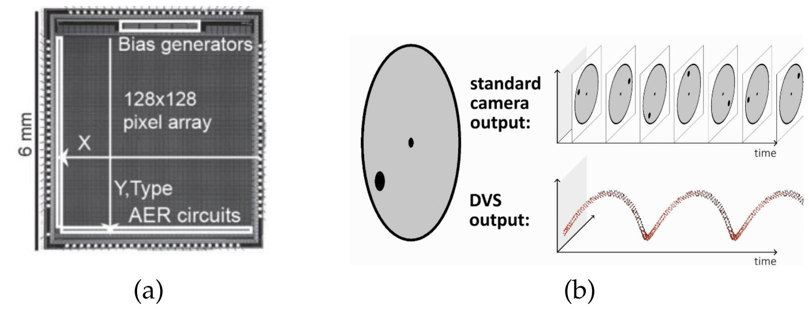
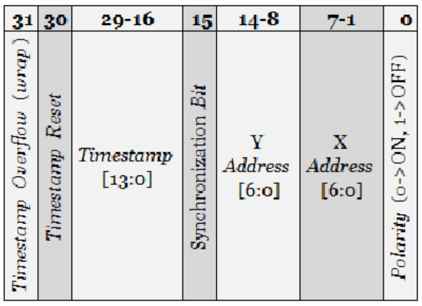
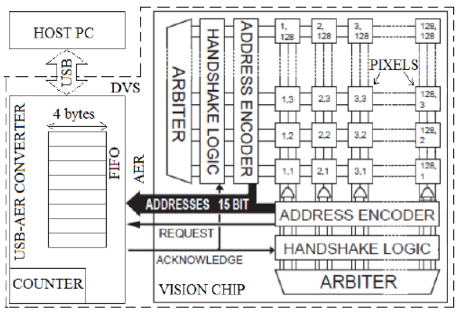


Figure 2.6: (a) 128x128 pixel array of a 4M2P DVS (b) Rotating disc stimulus output comparison between legacy vision sensors and DVS.

Event processing on no redundant information is performed in software within a micro controller or in FPGA and must be fast enough in order to meet real-time constraints. For example, for an average event rate of 200 kHz, each event must be processed in no more than 5 us, and just to give practical perception, this time interval is sufficient to balance a pencil on its tip. In [14], the average event rate is of 20 keps, and measurements show that 60 us are required to run the code for processing buffers of 128 events. This last number comes from the fact that events are stored as 4 bytes within the FIFO, and that the used USB 2.0 bulk transfer allows to transfer 512 bytes for high-speed endpoints, thus 512/4 results in a packet of 128 events.



31	30	29-16	15	14-8	7-1	0
Timestamp Overflow (wrap)	*Timestamp Reset*	*Timestamp* [13:0]	*Synchronization Bit*	Y *Address* [6:0]	X *Address* [6:0]	*Polarity* (0->ON, 1->OFF)

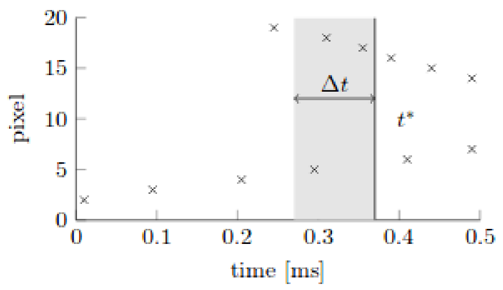


(a) (b)

Figure 2.7: (a) event sent through USB (b) DVS and HOST PC setup. Courtesy of [24].

Event based Computer Vision algorithms

Events directly produced by Dynamic Vision sensors cannot be processed to standard well known CV algorithms since a greyscale intensity image will not be produced in this case. Events are asynchronous and independent, so it is difficult to derive a general and complete understanding about the motion that characterizes the scene. The per se ambiguity that is associated with single events require to consider a set of particular past events as appropriate candidate to fulfill the event-based CV algorithm. The idea behind most of these algorithms is that of defining a particular set of past events in order to derive meaningful information to render the image so to apply the specific CV task. According to [57], there are two main standard approach used to define the set of past events used in order to render an image. The first approach consider a standard time interval over which events are accumulated, the problem arises for slow motion scene in which a larger time interval would be required in order to accumulate a satisfactory amount events, affecting latency. The second approach instead considers a fixed number of events, in this case the problem arises for heavily textured images that require an higher number of past events to be considered. Furthermore if two objects in the scene move at different speed neither a fixed time interval or number of events will represent a satisfactory strategy. In [56, 81] a new approach, based on the event lifetime concept, is proposed to overthrow these problems. The stand-alone ambiguous event will be associated to a finite temporal duration useful to represent the event over time. The aim is that of understanding for how much time the gradient that causes the events will remain at a certain pixel position. For each event, Mueggler et Al. compute the current velocity in terms of apparent motion in the image plane. This information is necessary in order to estimate the time that the moving edge requires to traverse 1 pixel. This approach overcomes the two main limitations previously exposed and permits to render sharp gradient images, i.e., edge like images, at any point in time. The mapping of event to lifetime allows to represent and show the event for its lifetime interval only, see Figure 2.8. This approach allows either to generate an edge like image at any time or to define a cluster for continuous objects tracking.



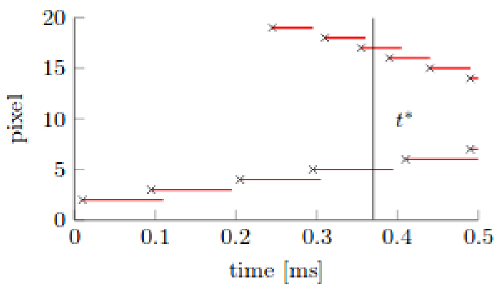


Figure 2.8: On the left events (crosses) are accumulated over a fixed time interval, on the right events are associated to their lifetime thus at time t* only active events are rendered, courtesy of [56].

When the lifetime of an event is computed, Mueggler et al in [56] augment the standard event with its lifetime τ and velocity v_x, v_y as follows:

$$< x, y, t, p > \mapsto < x, y, t, p, \tau, v_x, v_y >, \tag{2.6}$$

where the lifetime computed in funcion of $p = (x, y)^T$ is defined as:

$$\tau(\mathbf{p}) = \sqrt{{v_x}^{-2} + {v_y}^{-2}}, \tag{2.7}$$

and the velocities are computed starting from the first partial derivatives of $S(p) = (x, y, \sum_e(x, y)^T)$ where the last term in the vector represent the sum of active events at a certain time t:

$$\triangledown \sum_e (\mathbf{p}) = (\frac{\partial \sum_e}{\partial x}(p), \frac{\partial \sum_e}{\partial y}(p))^T. \tag{2.8}$$

The sum of active events is a monotonic increasing function and thus has no zero gradient for every possible **p**, which is related to velocities as follows:

$$\triangledown \sum_e (\mathbf{p}) = ((v_x^{-1}(\mathbf{p})), ((v_y)^{-1}(\mathbf{p})))^T. \tag{2.9}$$

2.2 Edge Computing

2.2.1 Preliminaries

The fifth generation (5G) mobile networks rely on software-defined networking (SDN) and network function virtualization (NFV) to support next generation services. Radio access and core functions are virtualized and executed in edge data centers according to the Multi-Access Edge Computing (MEC) principle. MEC is a key enabler for 5G mobile networks and was standardized by the European Telecommunications Standards Institute (ETSI) [31]. Formerly known as Mobile Edge Computing, MEC aims at providing computing service closer to the end user[1] by bringing applications and services at close distance to the end-user [80]. Thus, it finds applicability in scenarios where locality and low-latency are essential [77]. As its definition suggests, MEC is not tied to a single radio technology, but embraces both cellular and other radio access technologies such as WiFi. MEC is agnostic to the evolution of the mobile network itself, and can be deployed in LTE, 4G or 5G networks. The *edge*, also known as MEC host, is a data center or nano data center deployed close to the base stations inside an operator-owned infrastructure. The edge provides computing functionalities and can aggregate virtualized core and radio network functions of the mobile network. Such principle originates from the concept of cloudlets (CLs) [77] and enables resource-constrained mobile devices to prolong battery lifetime [22] while enhancing and augment performance of the mobile applications [48]. To this date,

[1]In the rest of the thesis, we will use the words *user*, *subscriber*, and *citizen* interchangeably

the research of edge computing has mainly focused on resource management and allocation by trading power consumption and communication delays [16, 93] while seminal works have mainly focused on the definition of architectural design principles [76, 53]. Emulation platforms for research in the area have only started to appear recently [2] and little attention has been paid to the problem of resource deployment. While in small networks the problem of cloudlet placement is not relevant because latency of user-to-cloudlet communications is negligible, it assumes a remarkable role in metropolitan areas, where a large number of users needs low-latency resources for many different applications, such as augmented reality or mobile gaming. Resource deployment is particularly interestingly and challenging in the context of smart cities and to the best of our knowledge only a vision paper has worked in the area by assessing the feasibility of leveraging three different infrastructures, i.e., cellular base stations, routers and street lamps and analyze the potential city coverage if only a subset of these elements is upgraded to furnish cloudlet capabilities [29]. This study is a important step forward to solve the problems of coverage, cloudlet selection and user-to-cloudlet assignment. However, it fails to fully capture the mobility dynamics of a city. Characterize the traffic dynamics with respect to different mobile services is crucial in order to dimension and manage the mobile network. In particular, investigating temporal behaviors and spatial patterns of citizens according to usage of different mobile services assumes a paramount importance. Recent studies unveil a consistent influence of urbanization level in the average traffic volume per-user by highlighting that different mobile services typically present different temporal behaviors but spatial patterns result uniform [50]. In this context, understanding the influence of citizens' spatial distribution and mobility on the consumption of mobile services is a promising direction to properly dimension and manage computational resources of mobile network.

2.2.2 Edge Computing for Computer Vision

Running computer vision algorithms on images or video collected by mobile devices represent a new class of latency-sensitive applications that expect to benefit from the edge cloud computing [94]. State-of-the-art AR/MR systems can almost correctly perceive the 3D geometry of the surrondings but they have not the same ability in terms of object detection and classification, however Convolutional Neural Networks (CNN) can be exploited to overcome these issues, but offloading computation to the edge represents a valid support since it is difficult to run large CNNs on mobile devices with low computational capacity [46]. In [30] it is presented a feature tracker that exploits both frame images collected by a legacy vision sensor and events collected by an Event camera. Their method extracts features on frames that by nature are not motion dependent and then they use events in order to track their features leveraging the low latency that caracterize these novel sensors. It may not be possible to associate the asynchronous events to frames on a mobile device with limited resources in terms of power, bandwidth, computation and heat dissipation, because of the heavy impact produced by the processing required for every frame, thus offloading computation to the edge would represent a valid possibility even if strict timings constraints must be carefully met both in terms of end to end system latency and in accuracy of detection or tracking. Figure 2.9 presents a latency diagram for a proposed event-based system that offloads CV tasks to the edge datacenter in order to send a feedback to the final user. DVS are able to detect a brightness change in the source scene with $\approx 1\ \mu$ s time resolution and to trasmit the event off chip in less than $\approx 15\ \mu$ s time. Events are then organized in packets and offloaded to the edge that will perform computing for the specific CV task, e.g. image tracking [52, 95, 26, 27]. As soon as the edge DC has performed its computations, it can send a feedback to the AR/MR device that will render the resulting image for the final user. Offloading CV tasks to the Edge represents a challenge for the on going research because of the strict latency requirements. More research attention has been given to mobile devices that locally perform CV task, but since the trend is changing and AR/MR devices are rapidly gaining attention it is necessary to consider the end to end latency requirements in these challenging cases. [61] and [70] both consider object detection offloading to the edge. The systems both require more than 400 ms offloading latency, which is not suitable for the correct rendering of objects in MR systems. In [46] a frame based system that reduces the offloading detection latency achieving high detection accuracy finishing the rendering too within 20 ms is proposed. The advantage would be of course higher if an event-camera would be considered since reality would not be captured at a constant frame rate but through asynchronous redundant free events.

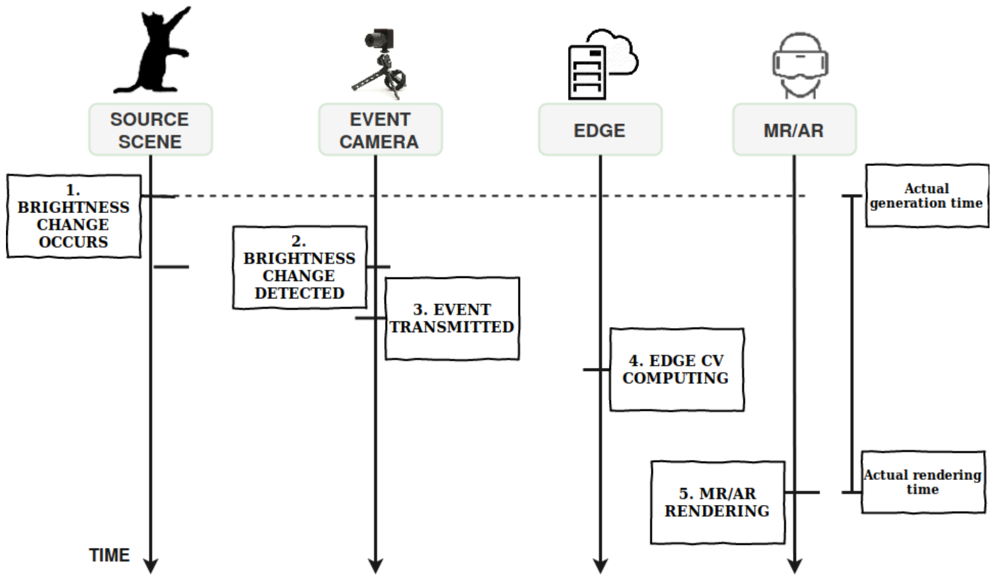


Figure 2.9: Event based system that offloads CV tasks to the edge.

Chapter 3

Measurements Framework

The chapter is organized as follows:

- Section 3.1 offers an overview of jAER, the open-source framework supporting event sensors that has been used for evaluations in the testbed proposed. It illustrates at first the GUI functioning and then, the communication socket set up for experiment implementation.

- Section 3.2 provides a preliminary analysis of some event camera datasets in terms of event rate considering different kind of motion.

- Section 3.3 presents the RTP Client-Server platform that has been used for evaluation of frame-based system.

- Section 3.4 illustrates CPU profiling techniques and describe the VisualVM profiling tool used.

3.1 The jAER Framework

3.1.1 Overview on jAER

The Java-based framework, jAER, consists of around 300 classes and allows both real time and offline processing of events generated by an event sensor. An Address Event (AE) sensor can be plugged in through a USB interface, and jAERViewer, see Figure 3.1, can be used in order to view, log, playback and process the events coming from the device [23, 84, 13]. The considered datasets have been obtained through DVS sensors that asynchronously collect and timestamp events with 1 us precision. Events are thus transmitted in a packet with variable size through USB to a host running jAER. Whenever a packet is received by jAER, a set of desired ordered filters is applied in order to modify and filter events as needed, see Figure 3.2. Event rendering can be performed according to a fixed time slice or number of events, visual annotations like cluster mass or identifier can be added to the output. In Figure 3.1, events collected by a flying drone within a office are rendered through the jAERViewer [11]. The jAERViewer is characterized by an infobar, Figure 3.3, on the top, a display that allows to render the image from data and a data slider, see the bottom part of Figure 3.4. From the left to the right the infobar contains:

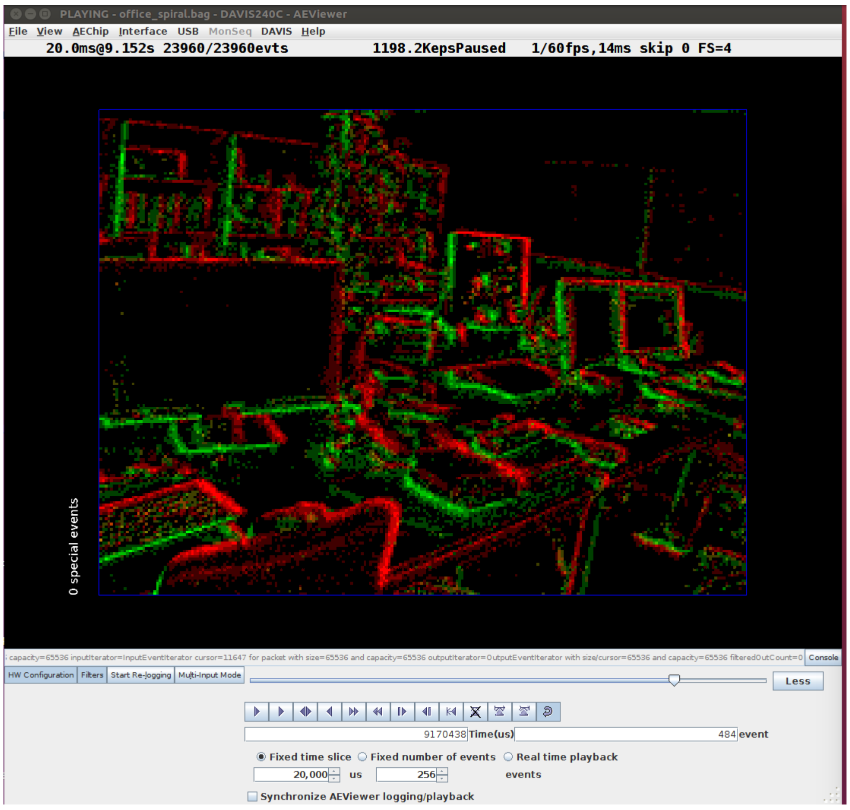
- *Time slice*: time interval covered by the frame that has been rendered.
- *Absolute time*: before the screen refresh, it represents the timestamp of the last event packet to be completely processed.
- *Raw events*: total number of events used to render the current frame.
- *Filtered events*: the filter or the set of filters applied, allow to eliminate some events, this field represents the total number of events after filtering.
- *Event rate*: represents the number of events generated per second where eps stands for event per second.
- *The play rate*: multiplication factor that allows to speed up the playback of a video with respect to the original one.
- *The actual frame rate and the desired frame rate*: the actual frame rate is the frame rate that is achieved by the viewer, that thus takes into account effective computational load and delay. The desired frame rate is the aimed one. In Figure 3.3, the desired frame rate has been set to 60 fps but only 29 fps are achieved due to the computational load.

- *The delay after a frame is rendered*: it gives an idea on the processor load since it defines the amount of time the process is idle before the main loop begins.
- *FS - Full (Colour) Scale*: it is a contrast control for the display, the higher the value of the FS, the higher the contrast.

The data slider of Figure 3.4 is only present when playback logged data. It informs the user on the playback position and it allows to increase or decrease the speed of the playback, to start or pause the recording. Different graphics option are available under the jAER framework. In Figure 3.1, ON events are represented in Green while OFF events are represented in red. However, different colors and display methods can be selected. For example, one can choose to represent events, considering the time dimension as well, thus according to their time generation. Newer events will be represented in dark grey and older events will be represented in clear grey. The filter panel is presented in the left of part of Figure 3.2. The filters available in jAER allow to perform different operations, e.g., extracting feature, tracking objects and showing information. They can be selected from the top menu of the jAERViewer by clicking the filter button. Whenever a set of events is received by jAER, a set of desired ordered filters is applied in order to modify and filter events as needed. In the set of experiments performed, three main filters have been considered:

- *Info filter*: this filter shows statistical information about the considered data stream, e.g., absolute time and date of the recording, the event rate of ON and OFF events.
- *Background activity filter*: this filter allows to remove background uncorrelated events, [1].
- *Rectangular Cluster Tracker*: this filter allows to perform object tracking by specifying different parameters like the number and the mass of the events within a cluster.

In the right panel of Figure 3.2, the User-Friendly-Control panel is presented. This interface allows to enable or disable the capture of frames and events, to adjust the event threshold and the balance between positive and negative events, as well as the pixel refractory period.



```
20.0ms@9.152s 23960/23960evts          1198.2KepsPaused   1/60fps,14ms skip 0 FS=4
```

```
; capacity=65536 inputIterator=InputEventIterator cursor=11647 for packet with size=65536 and capacity=65536 outputIterator=OutputEventIterator with size/cursor=65536 and capacity=65536 filteredOutCount=0
```

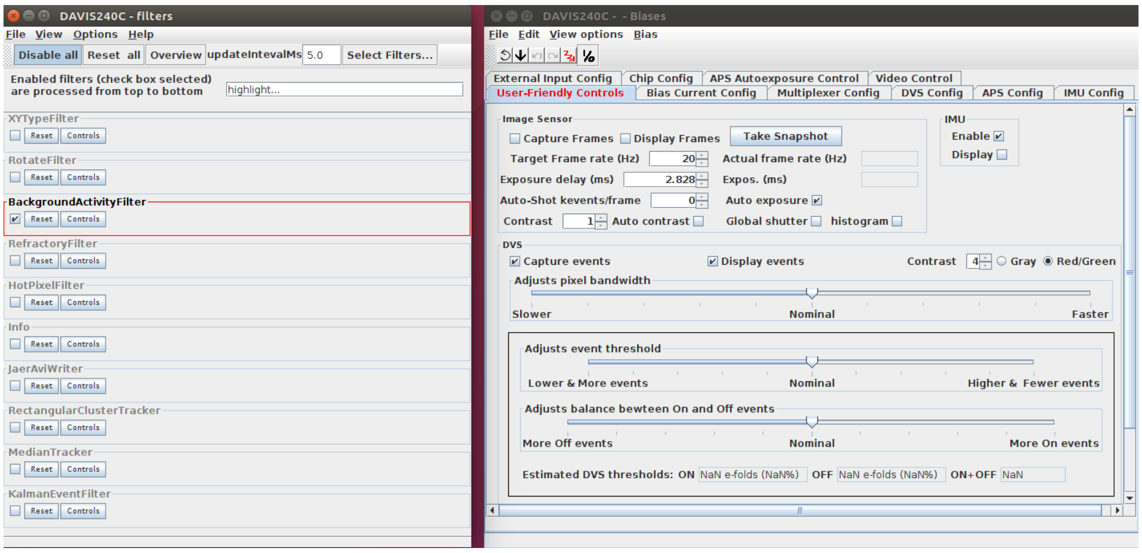
9170438 Time(us) 484 event

[✓ Fixed time slice] Fixed number of events Real time playback

20,000 us 256 events

☐ Synchronize AEViewer logging/playback

Figure 3.1: jAERViewer rendering events collected through a drone within an office.

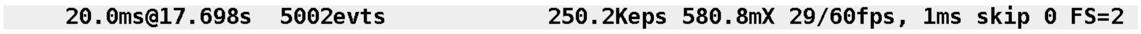


updateIntevalMs 5.0
Enabled filters (check box selected) are processed from top to bottom
highlight...

- [ ] XYTypeFilter
- [ ] RotateFilter
- [x] BackgroundActivityFilter
- [ ] RefractoryFilter
- [ ] HotPixelFilter
- [ ] Info
- [ ] JaerAviWriter
- [ ] RectangularClusterTracker
- [ ] MedianTracker
- [ ] KalmanEventFilter

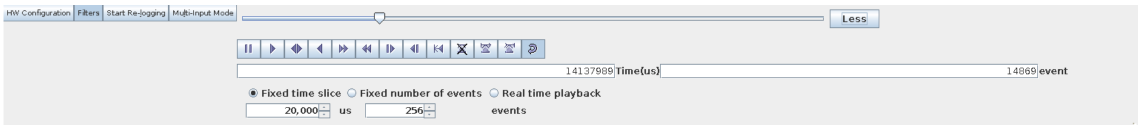
Image Sensor: Capture Frames ☐ Display Frames ☐
Target Frame rate (Hz) 20 — Actual frame rate (Hz)
Exposure delay (ms) 2.828 — Expos. (ms)
Auto-Shot kevents/frame 0 — Auto exposure ☑
Contrast 1 — Auto contrast ☐ — Global shutter ☐ — histogram ☐
IMU: Enable ☑ Display ☐
DVS: Capture events ☑ — Display events ☑ — Contrast 4 — Gray ○ [✓ Red/Green]
Adjusts pixel bandwidth
Adjusts event threshold
Adjusts balance bewteen On and Off events
Estimated DVS thresholds: ON NaN e-folds (NaN%) OFF NaN e-folds (NaN%) ON+OFF NaN

Figure 3.2: filter panel on the left and adjustable parameters on the right.



```
  20.0ms@17.698s  5002evts            250.2Keps 580.8mX 29/60fps, 1ms skip 0 FS=2
```

Figure 3.3: jAERViewer information bar.



14137989 Time(us) — 14869 event
[✓ Fixed time slice] ○ Fixed number of events ○ Real time playback
20,000 us — 256 events

Figure 3.4: jAERViewer dataslider.

3.1.2 Socket

A socket is an endpoint that allows to send and receive data within a node in a network. There exist two main types of Internet socket:

- *Datagram socket*: By using UDP, it provides a connectionless point for sending and receiving data. Packets sent over a Datagram socket are individually routed and delivered. In this case, there are not guarantees in terms of order and reliability, packets sent from an host to another may be delivered not in order and may not arrive at destination at all. Datagram Sockets is the Java mechanism for providing network communication via UDP instead of TCP [40].

- *Stream socket*: By using TCP, it provides a connection oriented point for sending and receiving data. The communication provided by a Stream socket is reliable and sequenced and with out-of-band capabilities. If delivery is impossible, the sender receives an error indicator.

Whenever live video streaming is considered, User Data Protocol, UDP, is recommended over Transport Control Protocol, TCP. In fact if TCP favors reliability, UDP instead favors reduced latency. The latter protocol does not wait for packet retransmission and acknowledgement. What matters for UDP, is not the frame loss, but the on-time delivery of the content. So, it results in the complete syncronization with live streaming [85].

The experiments have been performed over 802.11ac , the considered scenario is represented in Figure 3.5. The jAER framework allows to stream events from a Server, i.e., Dell Latitude E6430, to a Client, i.e., Dell Latitude E7440, using UDP. The jAER framework has been installed over both network nodes. For the experiment a WLAN with IP `192.168.1.0` and subnet `255.255.255.0` has been set. Static IP addresses have been assigned to the Server, i.e., `192.168.1.9` and the Client node, i.e., `192.168.1.10`. The Access Point is a NETGEAR Nighthawk X10 R9000 with `192.168.1.1` over the 5 GHz band.

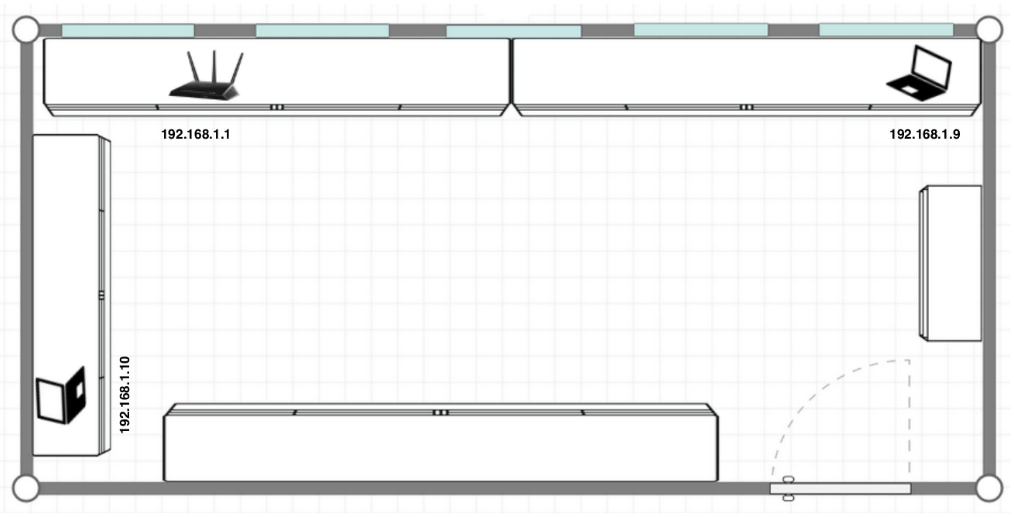
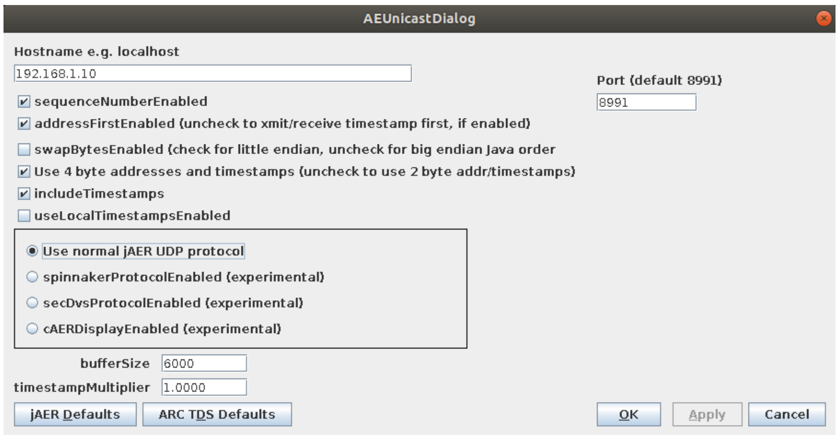


Figure 3.5: mmWave laboratory: The Server node, i.e., `192.168.1.9` streams event to the Client node, i,e., `192.168.1.10` through the AP, i.e., `192.168.1.1` over 802.11ac.

The UDP socket can be opened through the jAER File/Remote menu by selecting the Enable unicast datagram (UDP) output button for the Server,see Figure 3.6, and alongside the Open unicast (UDP) remote AE input for the Client. In order to set up the socket one needs to indicate, besides the IP address of the other node involved in the communication, the port over which the streaming will take place and the size of the buffer at Server and Client side, which according to results presented in Section4.2, has showed critical performance. By enabling the sequence number in the AEUnicastDialog, it is possible to keep track of the packet sent and this is useful for a number of purposes, including counting the number of losses. The address and timestamp couple can be represented over 2 Bytes or 4 Bytes (the latter as been used in the experiments). In the analysis performed the first `int32` of each AE is the address, and the second is the timestamp. Alternatively, it is possible to have the first `int32` as the timestamp, and the second as the address. As can be seen in Figure 3.6, different experimental protocols can be selected to perform the streaming, by the way standard UDP has been used in order to obtain results more general as possible.


AEUnicastDialog

Hostname e.g. localhost: 192.168.1.10

Port (default 8991): 8991

- ☑ sequenceNumberEnabled
- ☑ addressFirstEnabled (uncheck to xmit/receive timestamp first, if enabled)
- ☐ swapBytesEnabled (check for little endian, uncheck for big endian Java order
- ☑ Use 4 byte addresses and timestamps (uncheck to use 2 byte addr/timestamps)
- ☑ includeTimestamps
- ☐ useLocalTimestampsEnabled

- ● Use normal jAER UDP protocol
- ○ spinnakerProtocolEnabled (experimental)
- ○ secDvsProtocolEnabled (experimental)
- ○ cAERDisplayEnabled (experimental)

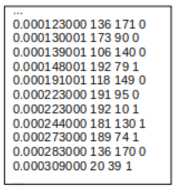
bufferSize: 6000

timestampMultiplier: 1.0000

Figure 3.6: Socket opened at the Server side towards the Client.

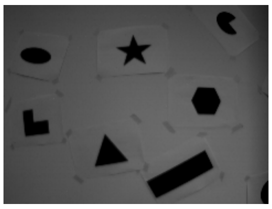
3.2 Dataset Analysis

In first instance, some of the event camera datasets for high speed robotics provided by [86] have been analyzed, see Figure 3.7 and Table 3.1. For each considered dataset, the text file *events.txt* which contains one event per row in the form of *timestamp*, *x* , *y*, *polarity*, has been logged at first through jAER and then processed through Matlab. The Cumulative Distribution Function for various datasets is presented in Figures 3.9 - 3.13. The computation has been performed locally on a Dell Latitude E6430 with an Intel $i7 - 3720QM$ processor at 2.60 GHz, 16 GB RAM and Linux Ubuntu 18.04 LTS, with the aim of understanding scene dynamicity and evolution in time. The streaming has been performed locally over the same host opening a socket through two instances of the jAERViewer, see Section 3.1.2. with a buffer size of 63000 Bytes both at transmitter and receiver side. The amount of motion in a scene, so the amount of events generated along time by an event-camera, depends on the considered scenario. The number of asynchronous events that are triggered by an event-camera is related to the scene complexity and to the speed of the moving objects.

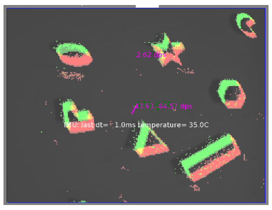


```
...
0.000123000 136 171 0
0.000130001 173 90 0
0.000139001 106 140 0
0.000148001 192 79 1
0.000191001 118 149 0
0.000223000 191 95 0
0.000223000 192 10 1
0.000244000 181 130 1
0.000273000 189 74 1
0.000283000 136 170 0
0.000309000 20 39 1
...
```

(a)



(b)



(c)

Figure 3.7: (a) events.txt file containing one event per row (b) Ground truth (c) jAER viewer events output: ON events in green and OFF events in red.

Table 3.1: Analysed datasets.

DATASET	MOTION	SPEED	CDF
1	Rotation	increasing	Fig.3.8
2	Translation	increasing	Fig.3.9
3	6-DOF	increasing	Fig. 3.10
4	Rotation	increasing	Fig.3.11
5	Translationn	increasing	Fig.3.12
6	6-DOF	increasing	Fig.3.13

Datasets have been generated with a DAVIS240C [19] for different kind of motion over 60 s. A higher number of events is generated as time increases because of the increasing speed, according to the event camera which is a motion detector by nature. The slope , thus the *event rate*, increases with increasing time since an higher number of events must be considered. In the different analyzed datasets various kind of motions are considered, i.e., rotation, translation and 6 Degree Of Freedom (DOF). In the rotation, translation and 6-DOF motion of the simple shapes dataset, Figure 3.8, 3.9 and 3.10, one can notice that most of the events are generated after 20 s or 15 s according to the increasing speed of the event camera moved by the operator in order to register the source video. Furthermore the 60 % of events is generated after 30 s , half of the total recording time. In the last 3 datasets, Figure 3.11, 3.12 and 3.13, the event-camera at the beginning has been moved at a lower speed with respect to the previous cases. In the first 20 s, less than the 20 % of the total events are generated.

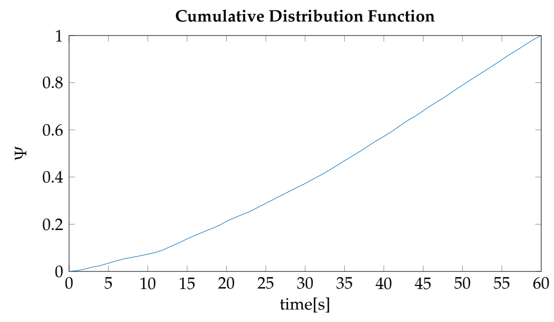


Figure 3.8: Simple shapes on wall rotation.

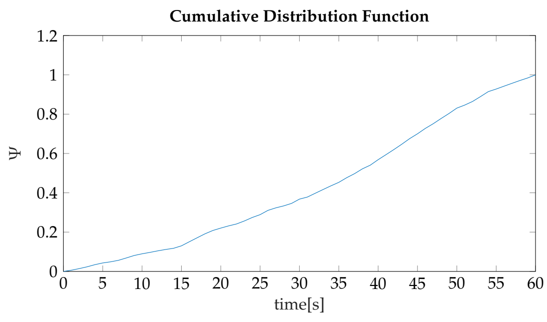


Figure 3.9: Simple shapes on wall translation.

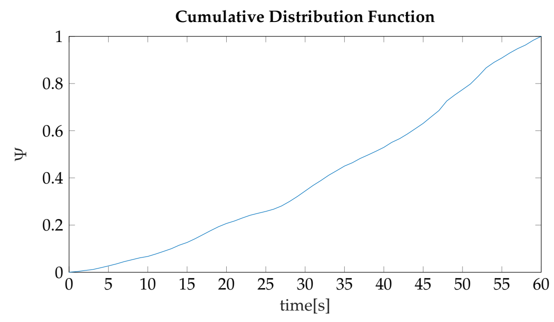


Figure 3.10: Simple shapes on wall 6– DOF.

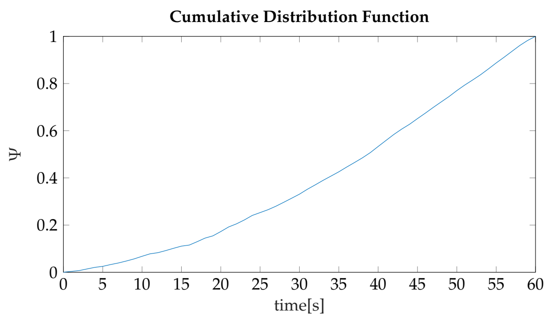


Figure 3.11: Poster Rotation.

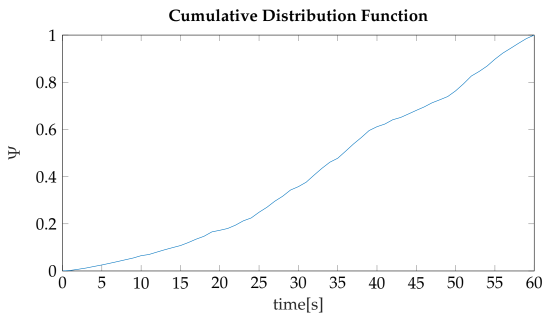


Figure 3.12: Poster 6-DOF.

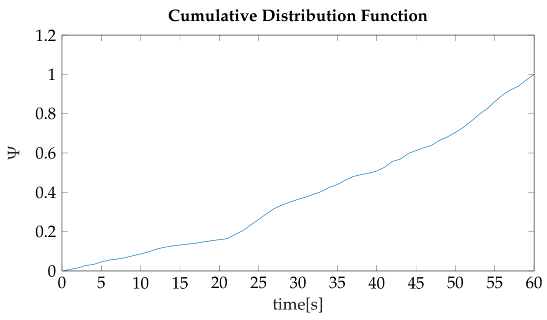


Figure 3.13: Poster translation.

3.3 RTP frame-based streaming video system

In order to understand the event-based streaming behavior with respect to the frame-based one, a custom frame-based Java framework has been developed starting from the code available at [60]. As Integrated Development Environment (IDE), Netbeans 8.2 has been used. Netbeans allows the use of a set of modular software components called modules in order to develop applications [62]. The streaming video Server and Client communicate using RTSP and send data using RTP, see Section 3.3. As in the event-based case, a datagram socket is created in order to send and receive UDP packets. In this case, the Client requests data to the Server, indicating the path to the desired video,i.e., the mp4 butterfly video created starting from .jpeg images, see Figure 2.3. Since an analysis also in terms of object tracking needs to be performed, the RTP Client has been opportunely modified. First of all a BlockingQueue has been created in order to decouple the process of receiving frames from the socket and the process of performing object tracking. As soon as the video is received by the RTP Client, it is decomposed to image frames. Image processing is performed on these extracted frames, separately. A BlockingQueue is a trade safe Java queue. It allows to perform flow control by blocking the queue if it is full or empty. Thus a thread cannot enqueues a new element if the queue is full and complementary cannot extract data from an empty queue [38, 39]. In order to avoid contention, two concurrent data structure can be used: an AtomicInteger and a Condition. The AtomicInteger allows to check the length of the queue, when the queue is in the contention-free state a signal notifies the waiting thread by modifying the Condition. A producer and a consumer thread have been developed, see Figure 3.14. The first thread is in charge of transferring data from the UDP socket to the BlockingQueue, the latter from the BlockingQueue to the GUI for rendering. In Figure 5.3, is represented the GUI at the Client side. The url field indicates the IP address of the Server, then the Port field represents the port number over which the communication takes place and the File field indicates the file that must be requested to the Server.

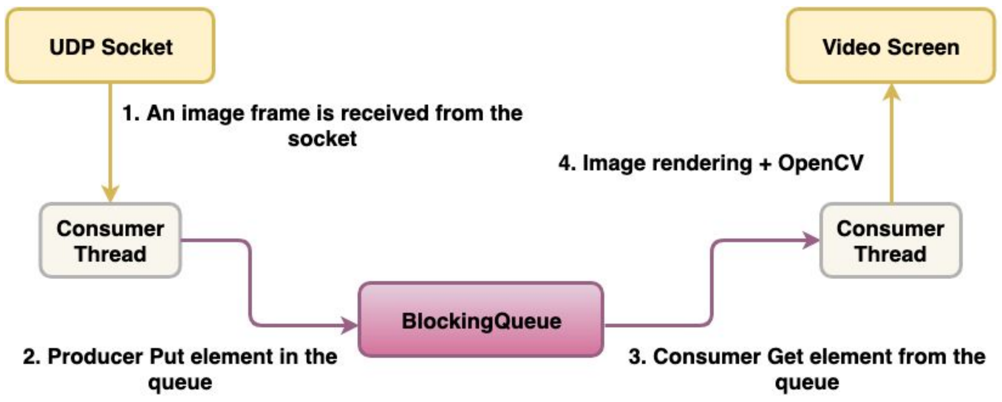

UDP Socket
1. An image frame is received from the socket
Consumer Thread
2. Producer Put element in the queue
BlockingQueue
3. Consumer Get element from the queue
Consumer Thread
4. Image rendering + OpenCV
Video Screen

Figure 3.14: Consumer-Producer data flow at the Client side.

3.4 Benchmark analysis

In computer programming, profiling is the dynamic analysis of a program behavior. It is performed in order to determine the part the target application that are executed most frequently or where most of the time is spent [33]. These portion of the code can be source code-level functions, e.g., subroutines and methods, basic blocks of code or even machine instructions [17]. By analyzing the performance of the target application it is possible to improve those sections of code that perform poorfly and may act as a bottleneck. The optimization of the CPU usage introduces several advantages. For example a faster user experience can be provided and device battery life can be preserved. In particular, through Java profiling, various JVM level parameters, like method or thread execution, object creation and garbage collection can be monitored. There exist two fundamental approach in Java profiling: “sampling” and “instrumentation”. The first periodically takes statistical data, the other performs integration into the code by setting an entry and an exit point.

Profiling by sampling

Sampling is an unintrusive profiling approach. In this case, no runtime modification of the target application occurs and the latter can run near to full speed. This approach is less numerical accurate since data are derived by a statistical approximation. Furthermore, since it is performed less often its overhead is negligible. The execution stack of the target application is probed at regular interval by the profiler through OS interrupts. In particular, at regular time slices, the OS interrupts the CPU, performing context switches. At this point, the profiler will take a sample, by recording the execution point. In this way, only fraction of the entire execution path are captured. Samples are then associated to the routine and source code to which they belong and a statistical measurement is performed in order to determine the frequency of critical parts within the run of the target application.

Profiling by instrumentation

With respect to the previous approach, this kind of profiler is intrusive. Custom byte code is added to the target application in order to record methods or threads execution as well as object creation. Probing has the advantage of providing greater flexibility at the expense of slowing down performance. Two main techniques can be considered. The first involves adding custom byte code to the source code, however this approach can create several confict problems. Furthermore, custom byte code can be added at best at the start of a procedure in source, missing important portions of execution time. The second approach, which works at runtime, is called binary profiling. Byte code is added to the executable code of the target once it is loaded in memory [25].

VisualVM profiling tool

VisualVM is a visual tool integrating commandline JDK tools and lightweight profiling capabilities that has been designed for both development and production time use [88, 89]. This tool as been used, see Section 3.4, in order to profile the performance of both event-based and frame-based scenarios,i.e., jAER and RTP Client/Server system. This open source and actively developed tool, allows to:

- *Display local and remote Java processes*: in the Applications left panel, there is a list of active local or remote Java applications.
- *Display process configuration and environment*: basic information such as PID or main class, JDK home or JVM for each involved process, are presented.
- *Monitor process performance and memory*: the dashboard shows CPU and heap usage, as well as number of loaded classes and threads.
- *Visualize process threads*: a timeline allows to display all active, running and waiting threads within the target application.
- *Profile performance and memory usage*: both sampling and instrumentation profilers can be used for performance analysis and memory management.

As mentioned before, both a statistical and instrumental profiler are available within VisualVM. In the first case, this tool takes a stack snapshot at regular intervals, recording the execution point. Finally, a statistical approximation is performed. In the second case instead, profiling custom code is injected within each method thus recompiling is required. By the way this kind of profiler has not been used since its implementation performs poorly.

Chapter 4

Analysis of Event-based Systems

This section presents the results when looking specifically to event-based systems. The next chapter will provide a comparative evaluation of event-based and frame-based systems. Several network experiments have been performed in order to evaluate the system behavior. As introduced in Section 3.1.2, the experiments have been performed at first over a LAN and then over 802.11ac. Events have been streamed, through a jAER socket, from a Server host, i.e., Dell Latitude E6430 running Ubuntu 18.04, to a Client, i.e., Dell Latitude E7440 with an Intel $i7-4600U$ processor at 2.1 GHz, 8 GB RAM and Linux Ubuntu 16.02. The jAER framework has been installed over both network nodes. For the experiment a WLAN with IP `192.168.1.0` and subnet `255.255.255.0` has been set. Static IP addresses have been assigned to the Server, i.e., `192.168.1.9` and the Client node, i.e., `192.168.1.10`. The Access Point is a NETGEAR Nighthawk X10 R9000 with `192.168.1.1` over the 5 GHz band, the IP address configuration has been maintained for the non-Wireless case too.

4.1 Scene complexity

Scene complexity and motion speed are determinant factors in the process that characterize the events formation. In the Wireless scenario, both for the Butterfly and Shapes dataset, Figure 2.3 and 2.4 respectively, the Cumulative Distribution Function of the UDP payload size for two very different buffer size values, i.e., 6000 Bytes and 63000 Bytes, has been evaluated. Figure 4.2 and 4.3 show the results. The maximum number of events max(e) for a UDP datagram over the jAER socket is given by:

$$\max(e) = \frac{\text{sizeof}(\mathit{buffer}) - \frac{\text{sizeof}(\mathit{integer})}{8}}{\text{sizeof}(\mathit{event})} \tag{4.1}$$

where the size of the buffer can be set through the socket panel, see Figure 3.6, the size of an integer is bound to 4 Bytes and that of an event to 8 Bytes. Thus, by increasing or decreasing the buffer size it is possible to increase or decrease the number of events fitting a UDP datagram, as well as the total number of packets sent. The Butterfly dataset is less complex with respect to the Shape one. In fact, the main motion in this case is given by the flying butterfly and by the flowers moved by the wind. In the other case instead, all the six simple shapes, represented over the wall, contribute in the process of generating events. A very different number of events is generated within the two considered datasets,i.e., 225876 events and 140992269 events, for the Butterfly and the Shapes dataset respectively. For a relatively small buffer size of 6000 Bytes, see Figure 4.2, the sample datasets have more or less the same behavior, with an average UDP payload size of around 2700 Bytes. For a larger buffer size the difference between the two source scenes becomes more evident, see Figure 4.3. In fact, it has been found that the average UDP payload size for the Butterfly dataset is 5400 Bytes, while for the Shape one this value grows up to 62900 Bytes. The reason behind this different behavior must be sought within the events generation process. In the Shape dataset, it is produced a number of events which is 624 times more than in the Butterfly dataset one. In the first dataset, an enormous amount of events is continuously generated, thus the available buffer size is filled up to its limit and delivered to the consumer thread that will send it over the UDP socket, see Figure 4.1. In the other case, instead, a smaller number of events is generated and the generation of events is not as continuous as the previous one. Thus, in order to do not affect latency the buffer is not filled up to its limit but smaller UDP datagrams are created and send. Figure 4.4 for both datasets, shows the total number of packets sent as a function of the buffer size. The same configuration of the WLAN as before, has been kept in this case too, with events streamed from the

Server to the Client host. As the buffer size increases, fewer and bigger packets are sent.

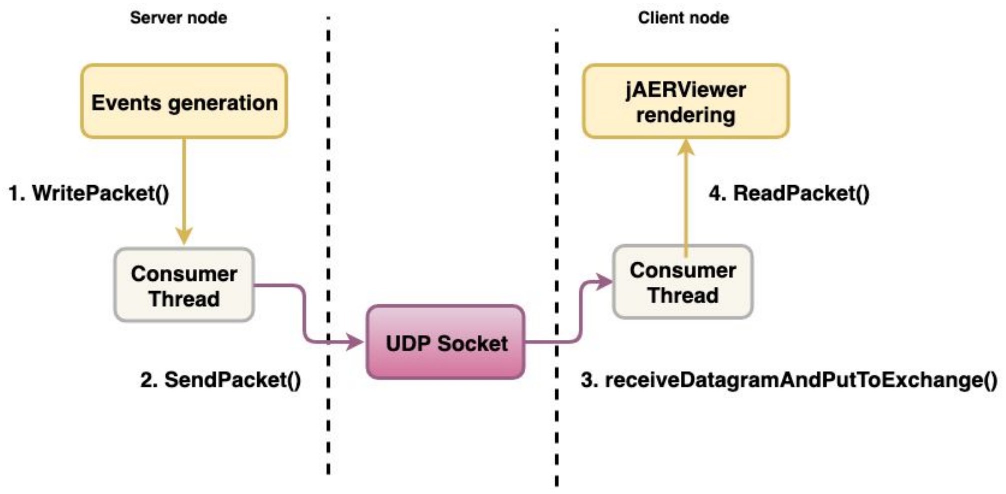


Figure 4.1: Streaming of events over jAER UDP socket.

A significant slope variation can be appreciated for low value of buffer size. In particular, in the Butterfly dataset, see bottom subplot of Figure 4.4, the total number of packets sent ranges from a maximum of 918 to a minimum of 260, for a buffer size that varies from 500 Bytes to 63000 Bytes. For the Shape dataset, upper subplot of Figure 4.4, the slope behavior is the same as the previous one, but the total number of packets sent is higher, according to the scene. In the latter case the total number of packets sent ranges from a maximum of 491084 to a minimum of 19460 whenever the buffer size ranges from 500 Bytes to 63000 Bytes.

4.2 The impact of packet losses

This subsection presents the results in terms of losses, both for packets and events, when data are streamed from the Server to the Client. Experiments were performed both in a Wireless and not-Wireless scenario, considering 802.11ac and Ethernet LAN. The usual IP configuration of the network nodes has been kept for this analysis, see Section 3.3.

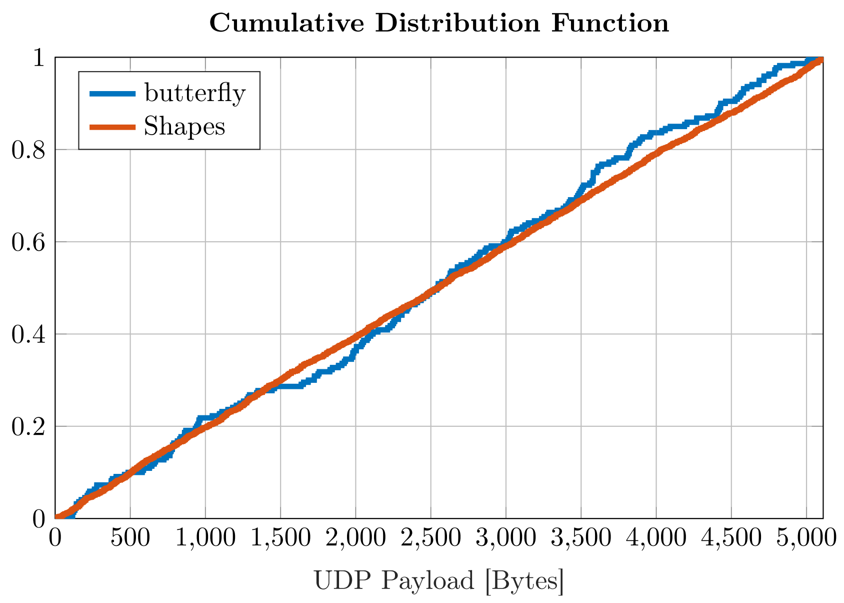


Figure 4.2: CDF of UDP payload for a buffersize of 6000 Bytes.

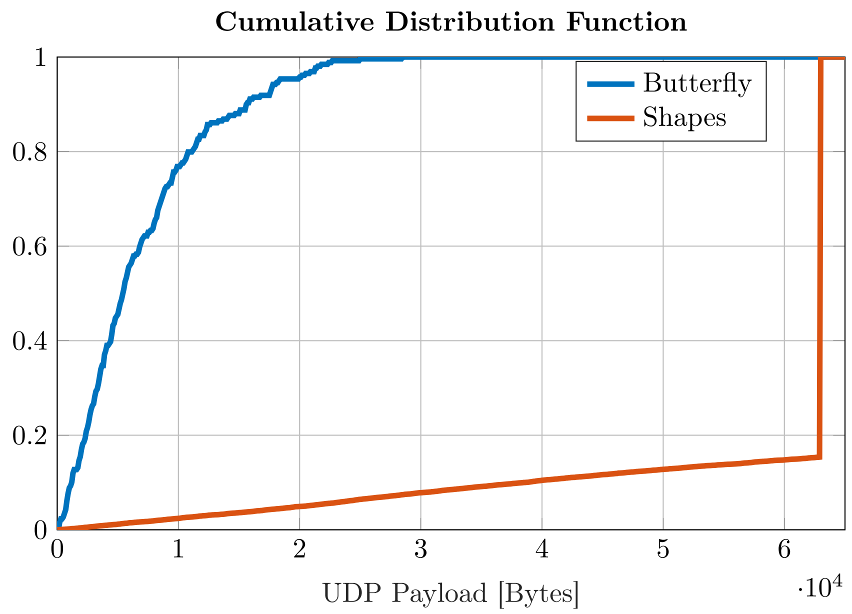


Figure 4.3: CDF of UDP payload for a buffersize of 63000 Bytes.

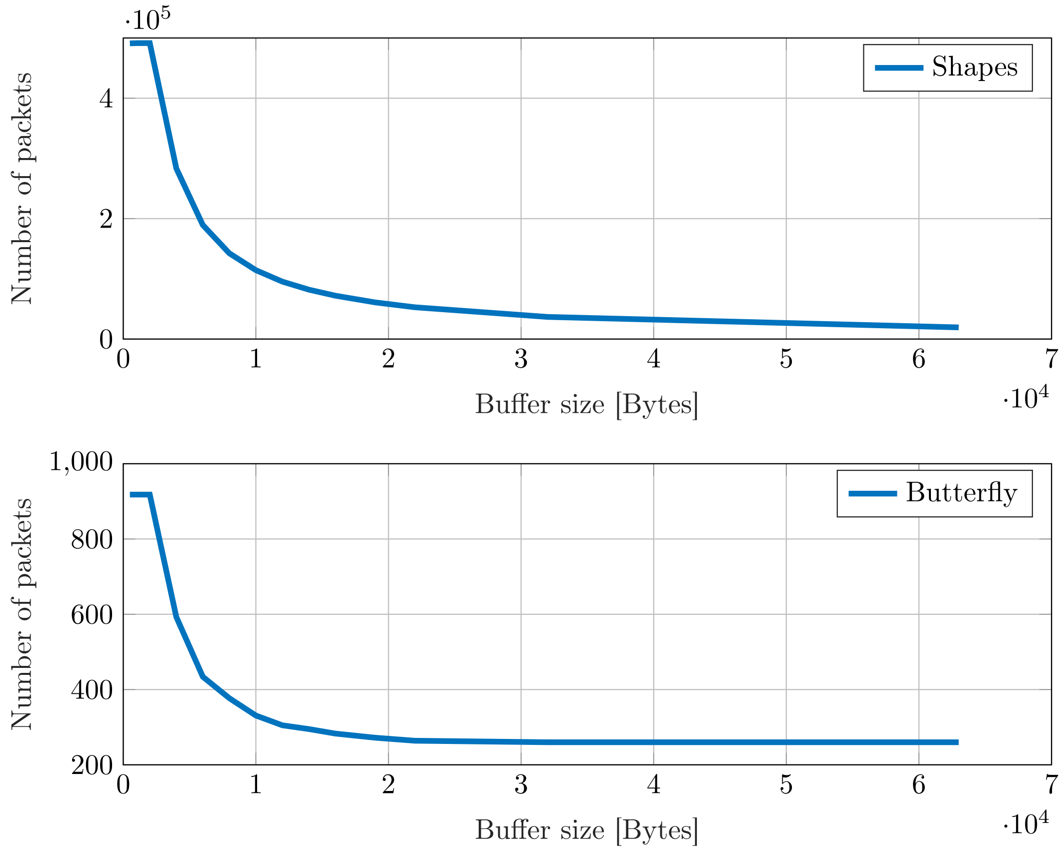


Figure 4.4: Number of packets sent for increasing buffer size.

4.2.1 Wireless Channels

The choice of the buffer size at the jAER application layer becomes crucial whenever packets are streamed over the network. Results show that a too small buffer size is detrimental for system performance since losses become predominant. A UDP datagram is filled with events by a producer thread and stored in the buffer, the datagram is then read by a consumer and send over the UDP socket. Events are generated with microsecond resolution, in pseudo-simultaneity with intensity changes arised in the reality. As previously explained, scene complexity and motion speed are determinant factors in the process that characterizes the events formation. For small values of the buffer size the application is not able to drain the bulky stream of events, the buffer overflows dropping packets and events. Looking at Figure 4.7 and 4.6, three main interesting observations can be highlighted. First of all, for higher value of the buffer size, i.e., from 4000 Bytes, an important difference can be noticed between the Butterfly and the Shapes dataset. In the latter, the amount of losses in terms of events and packets is more or less the same. In the Butterfly dataset instead, the two considered percentages are quite different. The reason behind this behavior is the different datasets complexity. As opposed to the Butterfly dataset, in the Shapes one, a continuous stream of events is generated and the UDP datagram packet is filled with events up to the maximum, see equation 4.1, thus in this scenario, whenever a packet is lost, the packet contains on average the maximum number of events. The second interesting result is that the optimal buffer size is not always associated to the maximum one. In fact, if zero losses are registered for a buffer size of 63000 Bytes in the Butterfly dataset case, the same consideration cannot be performed in the other case. According to the experiments performed, for the Shapes dataset, it has been found that the optimal buffer value is 6000 Bytes. In this case it has been registered an average packet loss of 17.53 % and an average event loss of 16.78 %. There were not particular interference in the Wireless scenario considered, and experiments have been repeated several times. The main reason behind this behavior, is probably the fact that,both at transmitter and receiving side, a buffer size of 6000 Bytes offers the possibility to the writing and reading thread, to efficiently drain the stream of events from the buffer. Finally, ad mentioned above, small values of the buffer size are detrimental for the system performance. In the Butterfly dataset, it has been registered an average packet loss of 46.29 % and an average event loss of 86.7 % for a buffer size of 500 Bytes. In the Shapes dataset instead, slightly higher values have been registered, with an average packet loss of 91.5209 % and an average event loss of 86.7 % for the same buffer size. The reason behind this behavior are several. Probably some of the events are lost at the source side due to buffer overflow, others at the receiving side because the reading thread is not able to rapidly extract events from the received packet. Packets correctly delivered to the application contains few events. For the Butterfly dataset, if a buffer size of 1000 Bytes is used, then a 74.51 % and a 47.16 % of events and packets losses has been registered. This means that only the 25.49 % of the total transmitted events is used in order to render the various frames at the Client node. In particular, in this case, this means that over a total of 225876 events, 57570 events are used. For the butterfly dataset, that has been used in order to perform object tracking, see Section 5.2, this number results still sufficient to perform object tracking but can negatively affect the performance of an object recognition algorithm loosing accuracy.

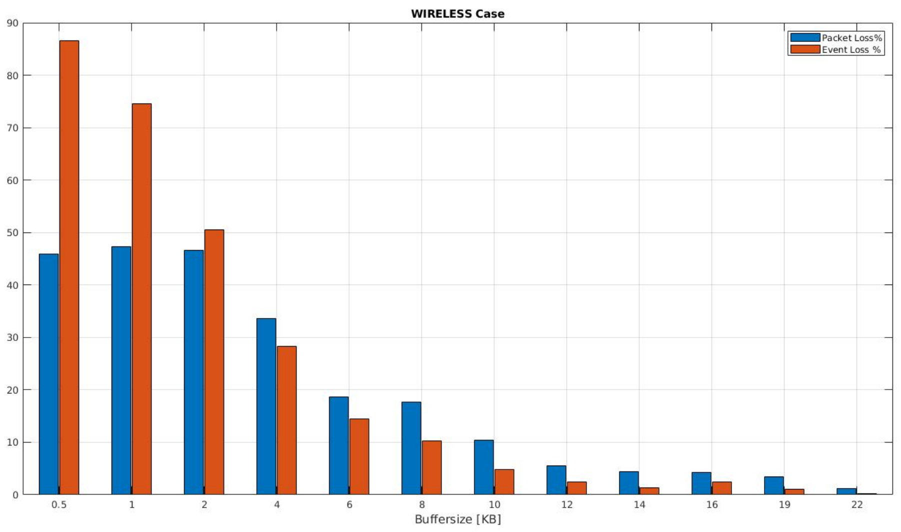


Figure 4.5: Butterfly dataset Wireless scenario.

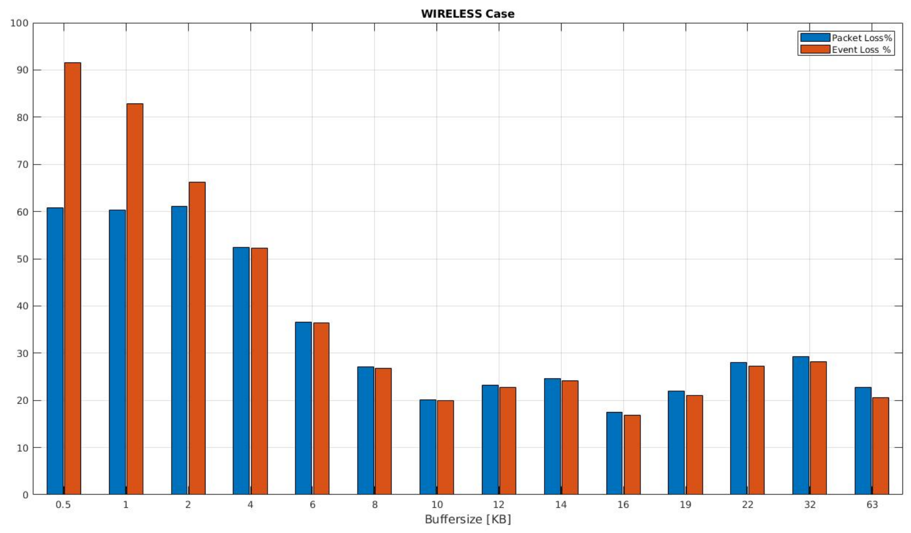


Figure 4.6: Shapes dataset Wireless scenario.

4.2.2 Non-Wireless Channels

The experiment has been repeated for the Butterfly dataset in the wired case too, with network nodes belonging to a LAN. The same considerations made for the Wireless case hold in this case too. For buffer size values greater than 4000 Bytes neither losses in terms of events or packets occur. For smaller values of the buffer size instead, a significant loss of events is registered,i.e., 75.80 % for a buffer set to 500 Bytes. The considerations made before are even more evident in this case. In fact, if 75.80 %, 52.8693 % and 12.15 % is the amount of event losses registered for 500 Bytes, 1000 Bytes and 2000 Bytes respectively, negligible packet losses are observed in these cases. This means that all packet received contain very few events with respect to the total available in the source dataset.

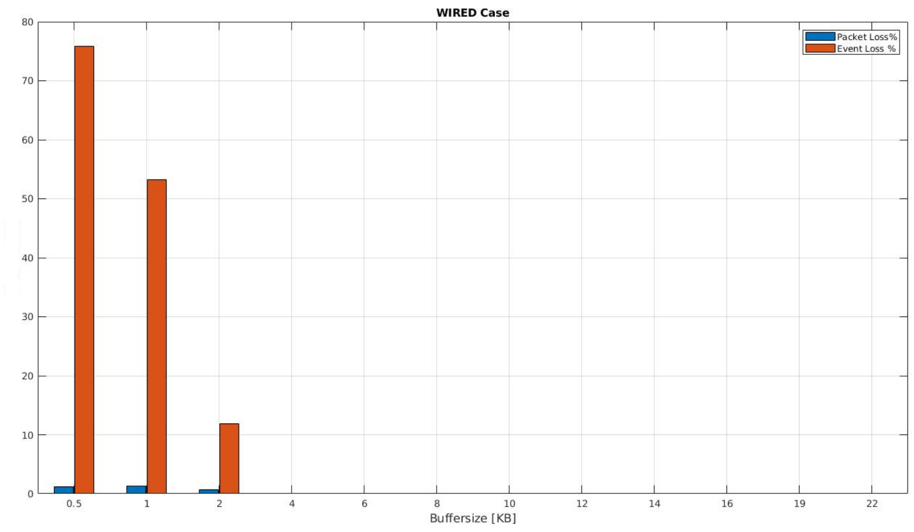


Figure 4.7: Butterfly dataset Non-Wireless scenario.

Chapter 5

Comparison of Event-based and Frame-based Systems

5.1 Throughput analysis

In this section the throughput for both the frame-based and event-based streaming has been evaluated when streaming the Butterfly dataset from the Server to the Client node. The RTP Client-Server Java custom platform, see Section 3.1, and the jAER framework, see Section 3.3, have been used to set up the UDP socket in the two cases. Results have been obtained by processing the Wireshark capture through tshark. TShark is a network protocol analyzer that allows to capture packet data from a live network. Alternatively, it allows to read packets from a previously saved capture file [83]. The average throughput achieved is 6 Mbps in the frame-based case, and 4 Mbps in the event-based one. The aim of this section is not that of comparing the performance of the two systems in terms of maximum throughput, since this comparison would not be fair. In fact while in the event-based case just ON and OFF events are sent , in the frame-based approach one has to take into account the 3 colour components. The idea is instead that of comparing the different throughput behavior. In the frame-based case the throughput has a regular and constant behavior, see Figure 5.1. In the event-based case instead, the throughput has an ON/OFF behavior ranging from 2 Mbps to 5 Mbps, see Figure 5.2. The throughput thus, changes dynamically within the

60 s of the video streaming. The latter result is congruent with the nature of the system. Events are asynchronously generated whenever a log intensity change is registered at a pixel. While frames are generated and sent at a constant rate, the asynchrony of the events generation translates into throughput fluctuations.

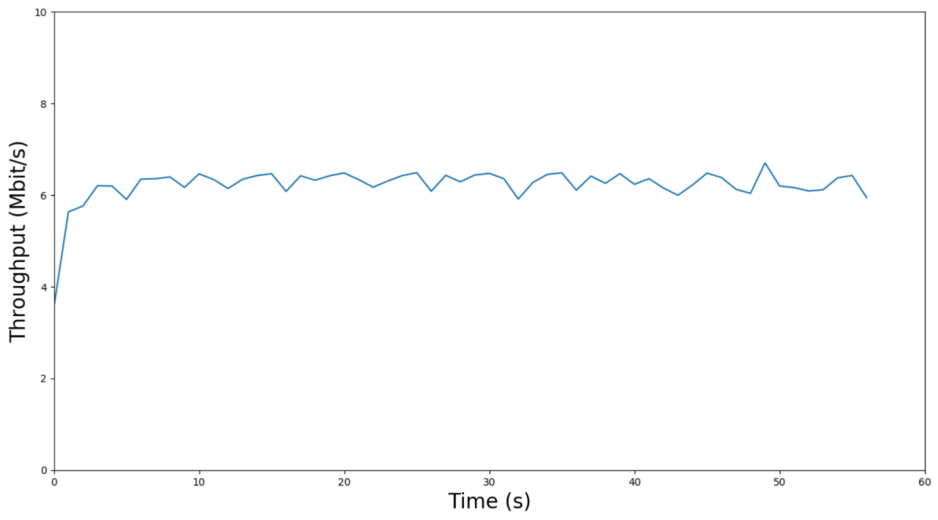


Figure 5.1: Frame-based streaming throughput

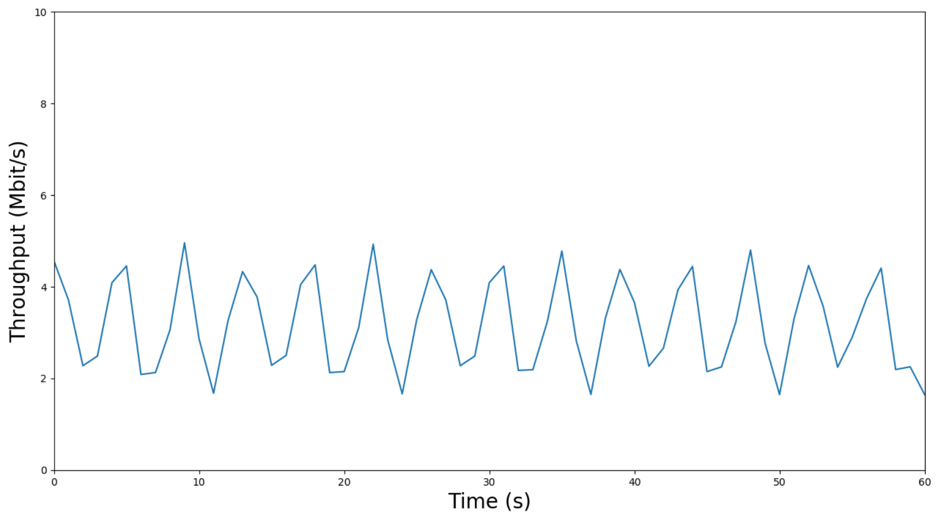


Figure 5.2: Event-based streaming throughput

5.2 Continuous object tracking

Augmented Reality is one core enabling technology of Industry 4.0. The aim behind this technology is that of improving user experience and productiveness, by completely change the way of working. Object tracking, as well as object recognition, will be fundamental actors within this Industry revolution, enabling the possibility of performing innovative tasks. These applications range from offering support in the design and production process to remote and automated assistance. Continuous object tracking has been performed both for the event-based and the frame-based case, with the aim of tracking the butterfly. For the sake of simplicity, this dataset has been chosen since suitable for object tracking. In fact, the background is static and uniform, while motion is given by the flying butterfly, i.e., the target, and the petals of the flowers.

5.2.1 Frame-based object tracking

The OpenCV Java-based algorithm available at [78] has been adapted to the frame-based case. As mentioned before, see Section 3.3, a consumer thread is in charge of transferring packet from the BlockingQueue to the GUI in order to perform object motion tracking. A basic algorithm, not optimized for the case has been used, since the aim is that of adding computation over the end point and to perform CPU profiling.The algorithm works by substraction. It compare two subsequent images and determine the pixel position at which there exists a significant intensity change, if the set of pixel is contiguous then an object has moved within the scene. Several object tracking method are available in literature, i.e., Mean shift [51], Kalman Filter [66], KLT tracker [7]. In particular the algorithm performs contour based object tracking. This method can be used both for image and video tracking. When an object is detected, its contour are extracted [3]. Contours are used in order to estimate the centroid position, with centroids representing the target object,see Figure 5.3. The algorithm goes through the following step:

1. ***Native library are loaded:***
   ```
   System.load("/java/opencv4/libopencv_java410.dylib");
   ```

OpenCV library must be integrated in Netbeans IDE.

2. ***Video frames are extracted from the BlockingQueue:***
   `poll = jpegBlockingQueue.take();`
   as mentioned before, a consumer thread is in charge of extracting frame from the BlockingQueue and to perform image processing.

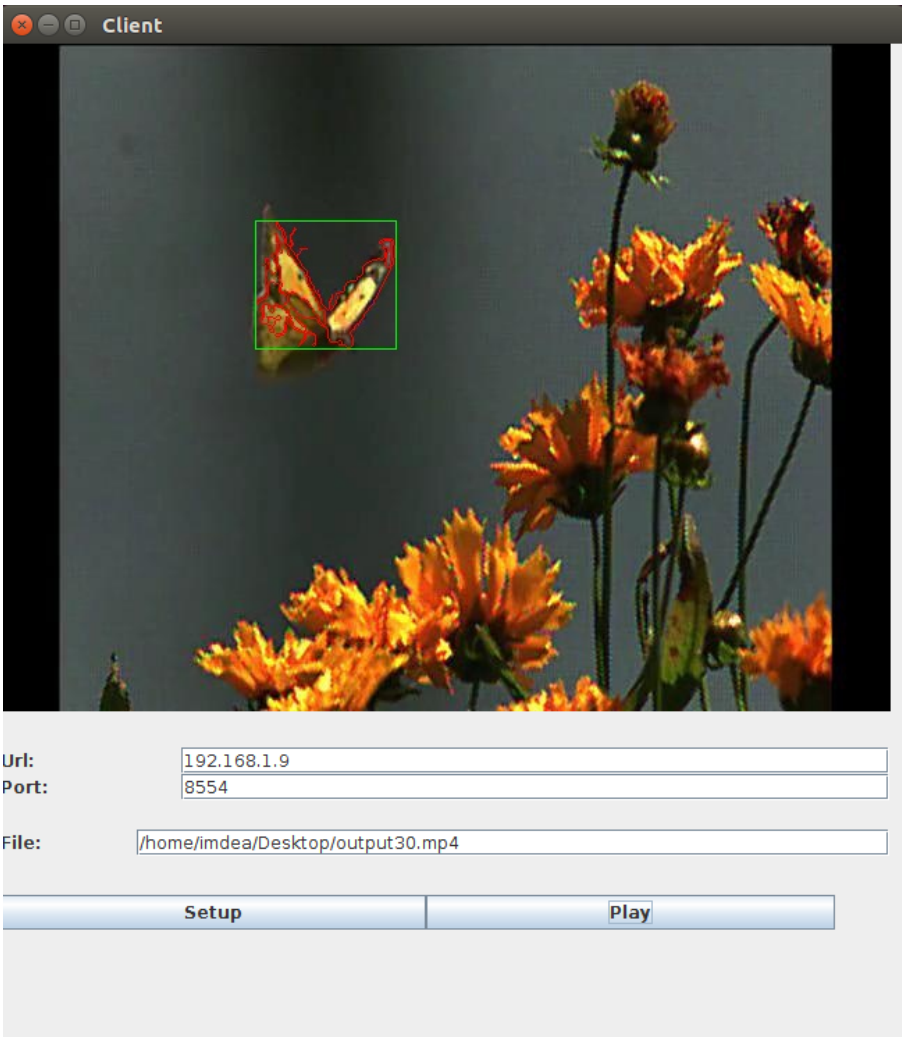
3. ***Conversion of the Mat structure into a Buffered image:***
   `Mat2bufferedImage(Mat image);.`
   the Java class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store several data like real and complex vectors or matrices, grayscale or color images [9]. The BufferedImage subclass instead, describes an image with an accessible image data buffer [8].

4. ***Conversion of the image from RGB to grayscale:***
   `Imgproc.cvtColor(frame, outerBox, Imgproc.COLOR_BGR2GRAY);`
   there exist more than 150 color-space conversion methods in OpenCV. However, two are the most widely used,i.e., from BGR to Gray scale and from BGR to HSV.

5. ***Successive image substraction:***
   `Core.subtract(outerBox, tempon_frame, diff_frame);`
   the Core class allows to perform matrix substraction operations.

6. ***Image binarization and adaptive threshold:***
   ```
   Imgproc.adaptiveThreshold(diff_frame, diff_frame, 255,
   Imgproc.ADAPTIVE_THRESH_MEAN_C,Imgproc.THRESH_BINARY_INV,
   5, 2);
   ```
   where the threshold value is the mean of the neighbourhood area.

7. ***Detection of homogeneous object:***
   `array = detection_contours(diff_frame);`
   detection of homogeneous objects is performed in order to determine the center of mass of each of them, by using the contour approach. Contours are detected for pixel values superior to an adjustable threshold.



Url: 192.168.1.9
Port: 8554
File: /home/imdea/Desktop/output30.mp4

Figure 5.3: Frame-based object tracking

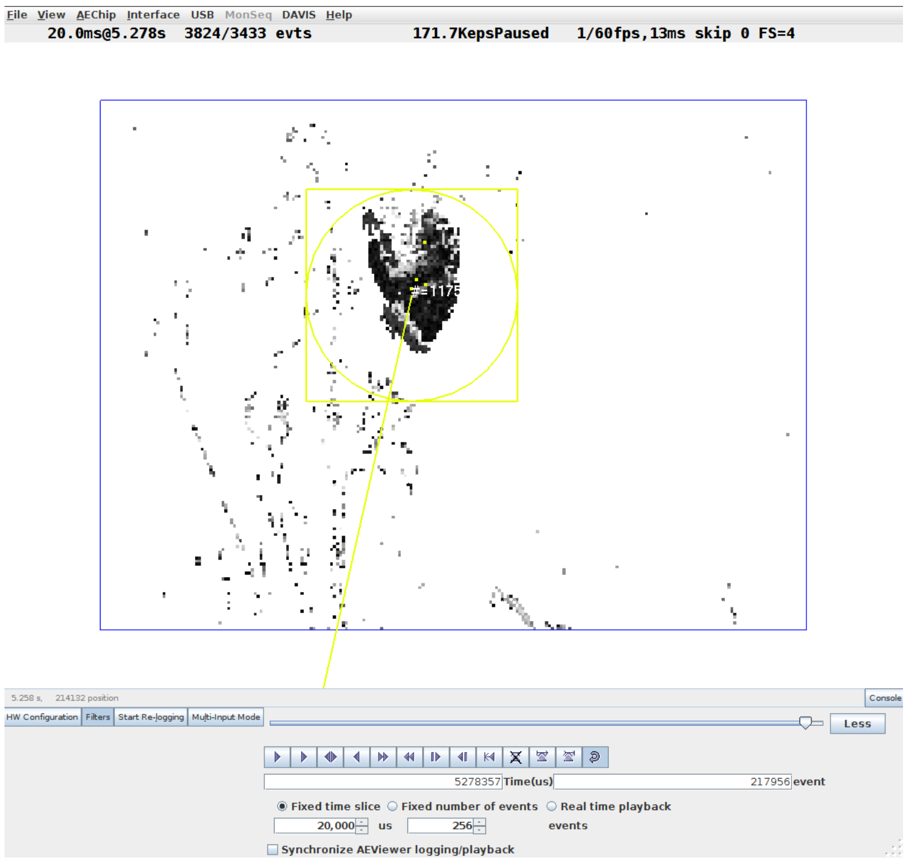
5.2.2 Event-based object tracking

In the event-based case, see Figure 5.4, in order to perform tracking, available jAER filters properly tuned have been used. In particular two Java class filters have been used:

- `BackgrondActivityFilter.java`.
- `RectangularClusterTracker.java`.

The first filter has been used in order to remove background uncorrelated events.

Events that pass the filter are the ones that are supported by another event in the past. Thus a correlation time,i.e., dt can be set. In this case only events that are correlated with a past event at the same location in within a 10 ms interval are allowed to be rendered. The RectangularClusterTracker has been exploited in order to perform object tracking. Several options are available for this filter. The number of clusters that aims at performing tracking has been set to one since only one moving target is considered and is is represented by the flying butterfly. The accuracy performance of the filter are not optimal since it has been designed for tracking a rectangular or square object. Nevertheless, as mentioned before the aim of this work is not that of optimizing the performance of a continuous object tracking algorithm. Instead, the objective is that of adding computation over the Client node in order to perform CPU and Java thread profiling, see Section 5.3.



```
20.0ms@5.278s  3824/3433 evts        171.7KepsPaused   1/60fps,13ms skip 0 FS=4
```

5.258 s, 214132 position

5278357 Time(us) 217956 event

[✓ Fixed time slice] Fixed number of events Real time playback

20,000 us 256 events

Synchronize AEViewer logging/playback

Figure 5.4: Event-based object tracking

In order to eliminate noisy events, the cluster lifetime parameter has been adjusted.

This parameter defines a cluster lasting time in µs. For the considered dataset, the cluster lasting time has been set to 10 µs . By decreasing this parameter the cluster life time is decreased. The cluster disappears if there are not sufficient incoming data to update its position. The size of the cluster has been modified according to the target dimension. This value represents the maximum distance from the cluster center to the event. The value, set to 0.06, is expressed as a fraction of maximum size of chip array. Since a DAVIS240C chip is used, a 240×180 array must be considered. The cluster has a size of $0,06 \times 240 = 14$ *pixels*. From Figure 5.4 one can notice that the mass of events is also associated to the yellow cluster, i.e., 1175. This value represents the number of events belonging to the cluster and can be used also to set a threshold. In this case only clusters with a mass greater then the defined threshold will be visible.

5.3 Benchmark profiling

VisualVm tool, see Section 3.4, has been used in order to analyse the performance in terms of CPU usage at Server and Client node for both the event-based and frame-based Java streaming platforms. Furthermore, both Server nodes extract data,i.e., events and frames in the two different cases, from a database. Whenever the VisualVM Java profiling is started for both the event-based and frame-based applications, 5 main threads, [35] can be observed:

- Attach Listener: this thread is started when the first attach request occurs.
- Signal Dispatcher: when the OS raises a signal to the JVM, this thread will pass the signal to the appropriate handler.
- Finalizer: this thread calls the finalizer methods.
- Reference Handler: this thread has high priority, it enqueues pending References. The GC creates a simple linked list of references which need to be processed and this thread quickly adds them to a proper queue and notifies ReferenceQueue listeners.

- DestroyJavaVM: this thread unloads the Java VM on program exit. Most of the time is in waiting state.

With respect to the RTP Server, the one running jAER creates UDP packets starting from events and renders events within the jAERViewer too. Whenever events are streamed from the Server node running jAER, the profiler report registers an average CPU usage of the 7 % , see Figure 5.5. This value is higher with respect to the frame-based case for which results show an average CPU usage of the 1.3 %, see Figure 5.6. According to the profiler results, there are two classes that use the 100 % of the CPU time. The first is the `AEViewer.Viewloop` class which involves the method for packet rendering, i.e., `Viewloop.renderPacket()`, and the second is the `AEUnicastOutput()`. The latter involves a consumer running thread,i.e., `Consumer.run()` that is in charge of sending AE packets.

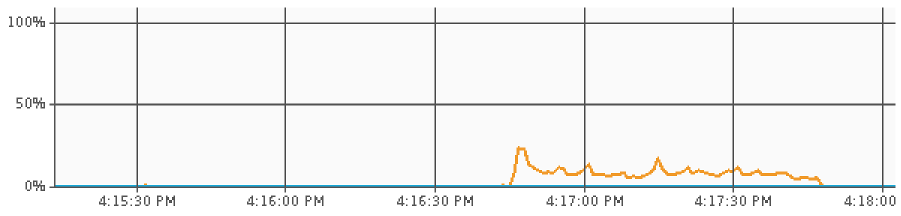


Figure 5.5: CPU usage for Server running jAER.

The RTP frame-based Server node shows a CPU usage really low, i.e., 1.3 %. As mentioned before, its implementation is based on a simple main thread and is relatively simple with respect to the jAER Server. According to the profiler results, the main thread use the 100 % of the CPU time. It involves a `getnextframe()` that loads data from the device in order to build the RTP packet, i.e., `RTP.packet()`.The RTP Server simple implementation sends a .jpeg image per UDP packet, with an average RTP payload size of 33193 Bytes.

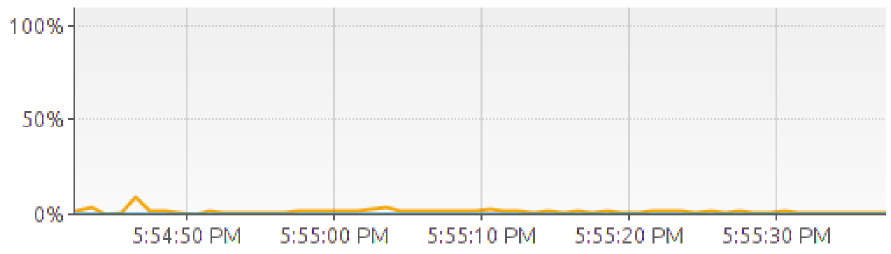


Figure 5.6: CPU usage for Server running RTPserver.

Whenever the Client node is considered the CPU usage is more or less the same in both cases. Results show an average CPU usage of 8.9 % for the Client node running jAER, see Figure 5.7 and a value of 10 % for the RTP Client, see Figure 5.8. It is important to highlight the fact that while the jAER framework accounts for a total of 300 classes, the RTP simple Client implemented instead, accounts for only 3 classes, i.e., `Main()`, `Producer()` and `Consumer()`. In particular, even if there is a strong difference in terms of complexity between the two considered applications, results show more or less the same behavior in terms of CPU usage. While the RTP Client node shows a constant CPU usage bounded to an average value of 10 %, the jAER one presents a particular fluctuating behavior. The main reason behind this behavior must be sougth in the different nature of the two involved systems. For the RTP Client, besides the key threads introduced at the beginning of this section, most of the CPU time is assigned to the `Producer()` that takes data from the socket and puts them in the BlockingQueue, and to the `Consumer()` thread that takes data from the BlockingQueue for rendering. In this case one frame is received more or less every 33 µs. For the jAER Client, events are received asynchronuosly whenever they are generated. This asynchrony translates into an alternate use of the CPU. As for the jAER Server node, most of the CPU time is used by the `AEViewer.Viewloop` in order to render events and by the `AEUnicastInput()` to perform packets processing and events extraction. In this case packet filtering for object tracking and background noise removal, i.e., `Filter.Packet()` takes approximately the 25.3% of the CPU time. In the frame-based system instead, image processing is performed by OpenCV that estimates how different is the actual frame with respect to the previous one. In general, an OpenCv algorithm for Single Input Multiple Data (SIMD), uses buffers of the Graphic card to parallelize the computation. These parts are not implemented in Java, but instead, in native C ++. The Java Native Interface (JNI) is then in charge to recall the C ++ native code. The native code is not executed on the Java Virtual Machine but directly from the OS. In this way VisualVM is not able to understand how many resources are used whenever a Java method calls a native function, thus another kind of profiler should be used.

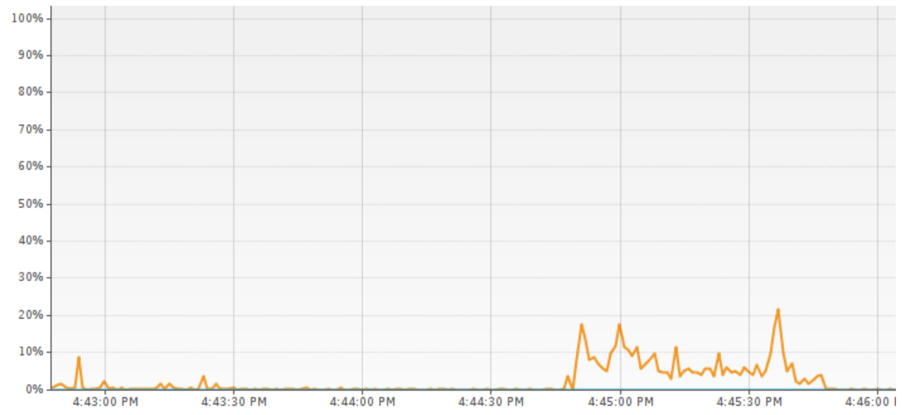


Figure 5.7: CPU usage for Client running jAER.

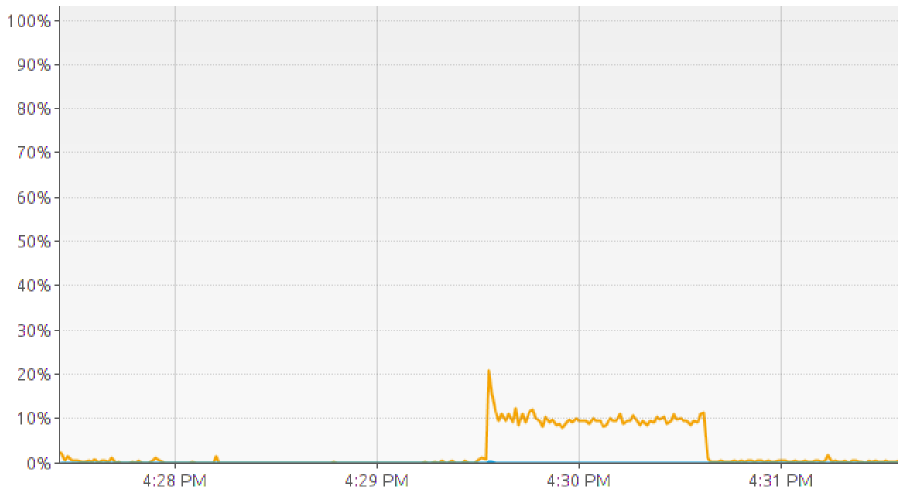


Figure 5.8: CPU usage for Client running RTPclient.

Chapter 6

Discussion and Future Research Directions

6.1 Edge Computing

In the past, more research attention has been given to mobile devices that locally perform CV tasks. This trend is changing. Nowadays, offloading major computer vision tasks to the edge datacenter represents an interesting possibility especially for mobile devices with limited computation capacity. As Augmented Reality (AR) and Mixed Reality (MR) systems keep increasing, supporting the revolution of the Industry 4.0, the offloading of computer vision tasks to the edge is gaining more and more popularity. By the way, the strict latency requirements of AR and MR applications represents a challenge for the on going research that aims at offloading CV tasks to the edge. For example, [61] and [70] consider object detection offloading to the edge. By the way, in their work, more than 400 ms offloading latency are required. This value is not suitable for the correct rendering of objects in MR systems. Several other approaches have been proposed in literature. In [94] they propose a latency-aware platform based on Apache storm. The proposed platform is able to support real-time responses with a per frame rendering latency of 32 ms. The system, proposed in [46] instead, enables high accuracy object detection for AR and MR systems at 60 fps, by introducing low latency offloading techniques. In [73], recurrent neural networks are used in order to reconstruct a video from single events over which off-the-shelf computer vision algorithms can be applied. It is undeniable that event-cameras because of their lightweight nature and of their pseudosimultaneity property with respect to reality changes, will find more and more space between video sensors whenever strict latency network requirements are fixed, e.g., Tactile Internet, section 6.2. Offloading the event stream computation to the edge datacenter will represent an interesting possibility whenever the network infrastructure will be able to support low latency communication delays.

6.2 Tactile Internet

Tactile internet, also know as *the Internet of skills*, comes with the promise of democratizing the access to skills and expertise for everyone, everytime and everywhere on Earth [34]. To concretize such vision, nowadays communication technologies require a strong push to enforce real-time perception of remote physical interactions of all five senses. Touch and kinesthetic movements pose particular challenges and require ultra-low latency to assure real-time remote control. Application domains of tactile internet stem from automotive, industry, health, teleoperations, to entertainment, gaming and virtual reality.

For tactile internet, the fifth generation (5G) mobile networks enforced ultra reliable and low latency communications (URLLC) [79]. The Third Generation Partnership Project (3GPP) has defined a target user plane latency of 0.5 ms for both uplink and downlink transmissions, and a reliability requirement of 99.999% to transmit a 32 Byte packet. Several solutions have been proposed so far, including a new waveform numerology (symbol length and subcarrier spacing), frame structure and reduction of transmit-time-interval (TTI), link adaptation strategies, semi-persistent scheduling, and grant-free access for uplink [41]. Besides network specific components, processing is becoming the bottleneck with the tight 1 ms round-trip latency-budget. In a sensor-to-actuator control loop, 0.65 ms are spent by the sensor, wireless and wired communication leaving only .35 ms for processing of data in a local multi-access edge computing (MEC) environment [92]. In Virtual/Augmented reality applications (AR/VR), tasks like object detection and recognition over one video frame can last more than 1 s if performed over mobile devices with TensorFlow Lite and convolutional neural networks [47]. Through intelligent decisions of which frames should be offloaded over the MEC servers and parallel processing [94], it is possible to meet AR/VR latencies of 100 ms. Frame skipping and preemption, i.e., interrupting the transmission of one frame in favor of another with more significant content, are other techniques used to reduce the amount of information to be transmitted and processed and achieve average glass-to-glass latencies of 21.2 ms, which are clearly above the tactile Internet latency budget of 1 ms.

Recently a new vision paradigm has emerged. Event-based cameras work radically different from traditional frame-based cameras because they measure brightness changes at a per-pixel granularity (i.e., events) rather than intensity measurements (i.e., frames) [68]. The high temporal resolution and low latency (events are transmitted once detected, at μs resolution much lower than the camera exposure time), and the high dynamic range (> 120 dB - high quality frame-based cameras achieve 60 dB) have made them the candidate instrument for robotic applications where real-time interaction is crucial, e.g., drone racing [15]. Other applications involve object tracking, recognition, gesture control, monitoring and surveillance [28]. To this date, event-based camera have always been employed on board or in close proximity of the device performing actuation while streaming events remotely remains an unexplored area.

Chapter 7

Conclusions

7.1 Summary of the Work

This thesis investigates the feasibility of a novel solution for the streaming of an event-based camera sensor data to a remote host. In fact, to this date, event-based camera have always been employed on board or in close proximity of the device performing actuation. This preliminary analysis is fundamental in order to understand how events are actually streamed over the network, in this way one can determine which are the network characteristics that must be guaranteed in order to successfully meet latency and accuracy.

This work proposes a measurement-based analysis of the streaming of event camera output, i.e., events, over 802.11ac, and it presents network-related results detailing the impact of events/packets losses and scene complexity.

The analysis has been performed through a Java-based framework, called jAER. This framework allows the visualization of real-time or recorded event-based data and the rapid development of real-time event-based algorithms and applications. Different datasets have been considered in order to perform a measurement evaluation able to take into account different kind of motion and speed as well as different textured scenes. Using jAER to perform extensive analysis, I investigated the issues related to the application framework. The results highlight that a too small buffer size is detrimental for latency and tracking performance because more than the 50% of events are dropped as the application threads involved are not able to keep up with events generation and packetization. I have implemented an RTP Java-based platform in order to compare the event and frame behavior within the network, considering in both cases continuous object tracking as CV task performed by the remote host.

Results show a fluctuating behavior both in the throughput and in the CPU usage. This behavior can be exploited to perform resources prediction allocation so to improve the performance of the wireless network.

The future development of event-based camera is strongly related to the growing opportunities in the emerging topics of neuromorphic perception and control. The pseudosimultaneity characteristic of these kind of sensors made them well suited for strict latency scenarios which involves autonomous driving and storm of drones coordination.

Bibliography

[1] *Adding a new event filter*. URL: https://sourceforge.net/p/jaer/wiki/Adding%20a%20new%20event%20filter/.

[2] Carlos Andrés Ramiro et al. "openLEON: An End-to-End Emulator from the Edge Data Center to the Mobile Users". In: *Proc. of ACM WiNTECH*. New Delhi, India, 2018, pp. 19–27. ISBN: 978-1-4503-5930-6. DOI: 10.1145/3267204.3267210.

[3] Jaya P Geethu Balakrishnan. "Contour based object tracking". In: *Red* 160.50 (), pp. 179–250.

[4] O. Blanco-Novoa et al. "A Practical Evaluation of Commercial Industrial Augmented Reality Systems in an Industry 4.0 Shipyard". In: *IEEE Access* 6 (2018).

[5] D. S. Bolme et al. "Visual object tracking using adaptive correlation filters". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2544–2550. DOI: 10.1109/CVPR.2010.5539960.

[6] Christos Bouras et al. "Real-Time Protocols (RTP/RTCP)". In: *Encyclopedia of Internet Technologies and Applications*. IGI Global, 2008, pp. 463–468.

[7] V. Buddubariki, S. G. Tulluri, and S. Mukherjee. "Multiple object tracking by improved KLT tracker over SURF features". In: *2015 Fifth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG)*. 2015, pp. 1–4.

[8] *Class BufferedImage*. URL: https://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImage.html.

[9] *Class Mat*. URL: `https://docs.opencv.org/java/2.4.9/org/opencv/core/Mat.html`.

[10] J. Conradt et al. "An embedded AER dynamic vision sensor for low-latency pole balancing". In: (2009), pp. 780–785. DOI: `10.1109/ICCVW.2009.5457625`.

[11] *Datasets*. URL: `http://rpg.ifi.uzh.ch/davis_data.html`.

[12] James Davis, Yi Hsuan Hsieh, and Hung-Chi Lee. "Humans perceive flicker artifacts at 500 Hz". In: *Scientific reports* 5 (Feb. 2015), p. 7861. DOI: `10.1038/srep07861`.

[13] Tobi Delbruck. "Frame-free dynamic digital vision". In: *Proceedings of the International Symposium on Secure-Life Electronics, Advanced Electronics for Quality Life and Society* (Mar. 2008). DOI: `10.5167/uzh-17620`.

[14] Tobi Delbruck and Manuel Lang. "Robotic Goalie with 3ms Reaction Time at 4 percent CPU Load Using Event-Based Dynamic Vision Sensor". In: *Frontiers in neuroscience* 7 (Nov. 2013), p. 223. DOI: `10.3389/fnins.2013.00223`.

[15] J. Delmerico et al. "Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset". In: *Proc. IEEE ICRA*. 2019, pp. 1–7.

[16] R. Deng et al. "Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption". In: *IEEE Internet of Things Journal* 3.6 (2016), pp. 1171–1181. ISSN: 2327-4662. DOI: `10.1109/JIOT.2016.2565516`.

[17] Mikhail A Dmitriev. *Hybrid profiling technique*. US Patent 7,519,959. 2009.

[18] *Documentation*. URL: `https://ffmpeg.org/documentation.html`.

[19] *Documentation*. URL: `https://inivation.com/support/hardware/davis240/`.

[20] Matthias Faessler et al. "Autonomous, vision-based flight and live dense 3D mapping with a quadrotor micro aerial vehicle". In: *Journal of Field Robotics* 33.4 (2016), pp. 431–450.

[21] Davide Falanga et al. "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5774–5781.

[22] C. Fiandrino et al. "Profiling Performance of Application Partitioning for Wearable Devices in Mobile Cloud and Fog Computing". In: *IEEE Access* 7 (2019), pp. 12156–12166. ISSN: 2169-3536. DOI: `10.1109/ACCESS.2019.2892508`.

[23] *Follow jaer*. URL: `https://sourceforge.net/projects/jaer/`.

[24] Jesus Armando Garcia Franco, Juan Luis del Valle Padilla, and Susana Ortega Cisneros. "Event-based image processing using a neuromorphic vision sensor". In: *2013 IEEE International Autumn Meeting on Power Electronics and Computing (ROPEC)* (2013), pp. 1–6.

[25] *Fundamentals of Performance Profiling*. URL: `https://smartbear.com/learn/code-profiling/fundamentals-of-performance-profiling/`.

[26] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. "A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3867–3876.

[27] Guillermo Gallego et al. "Event-based camera pose tracking using a generative event model". In: *arXiv preprint arXiv:1510.01972* (2015).

[28] Guillermo Gallego et al. "Event-based Vision: A Survey". In: *CoRR* abs/1904.08405 (2019). arXiv: `1904.08405`. URL: `http://arxiv.org/abs/1904.08405`.

[29] Julien Gedeon et al. "A Multi-Cloudlet Infrastructure for Future Smart Cities: An Empirical Study". In: *Proc. of 1st ACM International Workshop EdgeSys*. Munich, Germany, 2018, pp. 19–24. ISBN: 978-1-4503-5837-8. DOI: `10.1145/3213344.3213348`.

[30] Daniel Gehrig et al. “Asynchronous, photometric feature tracking using events and frames”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 750–765.

[31] Fabio Giust et al. *MEC Deployments in 4G and Evolution Towards 5G*. ETSI White Paper. 2018.

[32] D. Gorecky et al. “Human-machine-interaction in the industry 4.0 era”. In: *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. 2014, pp. 289–294. DOI: `10.1109/INDIN.2014.6945523`.

[33] Jason D Hibbeler and Jhy-Chun Wang. *Dynamic CPU usage profiling and function call tracing*. US Patent 7,093,234. 2006.

[34] O. Holland et al. “The IEEE 1918.1 “Tactile Internet” Standards Working Group and its Standards”. In: *Proceedings of the IEEE* 107.2 (2019), pp. 256–279. ISSN: 0018-9219. DOI: `10.1109/JPROC.2018.2885541`.

[35] *HotSpot Dynamic Attach Mechanism*. URL: `http://openjdk.java.net/groups/hotspot/docs/Serviceability.html`.

[36] *http://rpg.ifi.uzh.ch/davis_data.html*. URL: `http://rpg.ifi.uzh.ch/davis_data.html`.

[37] *Il protocollo di trasporto RTP*. URL: `http://www.ce.unipr.it/~corradi/livestreamer/RTP.htm`.

[38] *Interface BlockingQueue*. URL: `https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html`.

[39] *Interface BlockingQueue*. URL: `https://www.geeksforgeeks.org/blockingqueue-interface-in-java/`.

[40] *Java.net.DatagramSocket class in Java*. URL: `https://www.geeksforgeeks.org/java-net-datagramsocket-class-java/`.

[41] K. S. Kim et al. “Ultrareliable and Low-Latency Communication Techniques for Tactile Internet Services”. In: *Proceedings of the IEEE* 107.2 (2019), pp. 376–393. ISSN: 0018-9219. DOI: `10.1109/JPROC.2018.2868995`.

[42] B. Kueng et al. "Low-latency visual odometry using event-based feature tracks". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 16–23. DOI: `10.1109/IROS.2016.7758089`.

[43] *learnopencv*. URL: `https://www.learnopencv.com/`.

[44] W. Liang et al. "WIA-FA and Its Applications to Digital Factory: A Wireless Network Solution for Factory Automation". In: *Proceedings of the IEEE* (2019), pp. 1–21. ISSN: 0018-9219. DOI: `10.1109/JPROC.2019.2897627`.

[45] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. "A 128x128 120dB 15us latency asynchronous temporal contrast vision sensor". In: 43 (Jan. 2007), pp. 566–576.

[46] Luyang Liu, Hongyu Li, and Marco Gruteser. "Edge assisted real-time object detection for mobile augmented reality". In: (2019).

[47] Luyang Liu, Hongyu Li, and Marco Gruteser. "SkyCore: Moving Core to the Edge for Untethered and Reliable UAV-based LTE Networks". In: *Proc. ACM MobiCom*. Los Cabos, Mexico, 2019. ISBN: 978-1-4503-5903-0. DOI: `10.1145/3300061.3300116`.

[48] P. Mach and Z. Becvar. "Mobile Edge Computing: A Survey on Architecture and Computation Offloading". In: *IEEE Communications Surveys Tutorials* 19.3 (2017), pp. 1628–1656. ISSN: 1553-877X. DOI: `10.1109/COMST.2017.2682318`.

[49] Ana I. Maqueda et al. "Event-based Vision meets Deep Learning on Steering Prediction for Self-driving Cars". In: *CoRR* abs/1804.01310 (2018). arXiv: `1804.01310`. URL: `http://arxiv.org/abs/1804.01310`.

[50] Cristina Marquez et al. "Not all apps are created equal: Analysis of spatiotemporal heterogeneity in nationwide mobile service usage". In: *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM. 2017, pp. 180–186.

[51] *Mean shift Tracking*. URL: `http://www.cse.psu.edu/~rtc12/CSE598G/introMeanShift.pdf`.

[52] Anton Mitrokhin et al. "Event-based moving object detection and tracking". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–9.

[53] C. Mouradian et al. "A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges". In: *IEEE Communications Surveys Tutorials* 20.1 (2018), pp. 416–464. ISSN: 1553-877X. DOI: `10.1109/COMST.2017.2771153`.

[54] *MP4 file format uses , features , advantages and disadvantages*. URL: `https://www.online-sciences.com/technology/mp4-file-format-uses-features-advantages-and-disadvantages/`.

[55] *MPEG-4*. URL: `https://web.archive.org/web/20141031153948/https://images.apple.com/quicktime/pdf/MPEG4_v3.pdf`.

[56] E. Mueggler et al. "Lifetime estimation of events from Dynamic Vision Sensors". In: (2015), pp. 4874–4881. ISSN: 1050-4729. DOI: `10.1109/ICRA.2015.7139876`.

[57] Elias Mueggler. "Event-based Vision for High-Speed Robotics". In: (2017).

[58] Elias Mueggler, Basil Huber, and Davide Scaramuzza. "Event-based, 6-DOF pose tracking for high-speed maneuvers". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2014), pp. 2761–2768.

[59] *Multimedia Networking Applications*. URL: `https://www.net.t-labs.tu-berlin.de/teaching/computer_networking/06.01.htm`.

[60] *mutaphore/RTSP-Client-Server*. URL: `https://github.com/mutaphore/RTSP-Client-Server`.

[61] Saman Naderiparizi et al. "Glimpse: A Programmable Early-Discard Camera Architecture for Continuous Mobile Vision". In: *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '17. Niagara Falls, New York, USA: ACM, 2017, pp. 292–305. ISBN: 978-1-4503-4928-4. DOI: `10.1145/3081333.3081347`. URL: `http://doi.acm.org/10.1145/3081333.3081347`.

[62] *Netbeans: Documentation, Training and Support.* URL: `https://netbeans.org/kb/index.html`.

[63] *OpenCV-Doc.* URL: `https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/`.

[64] Jan Ozer. *Video compression for multimedia.* AP Professional, 1995.

[65] Volker Paelke. "Augmented reality in the smart factory: Supporting workers in an industry 4.0. environment". In: *Proceedings of the 2014 IEEE emerging technology and factory automation (ETFA)*. IEEE. 2014, pp. 1–4.

[66] Hitesh A Patel and Darshak G Thakore. "Moving object tracking using kalman filter". In: *International Journal of Computer Science and Mobile Computing* 2.4 (2013), pp. 326–332.

[67] J. A. Perez-Carrasco et al. "Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing–Application to Feedforward ConvNets". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11 (2013), pp. 2706–2719. ISSN: 0162-8828. DOI: `10.1109/TPAMI.2013.71`.

[68] C. Posch et al. "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output". In: *Proceedings of the IEEE* 102.10 (2014), pp. 1470–1484. ISSN: 0018-9219. DOI: `10.1109/JPROC.2014.2346153`.

[69] Kari Pulli et al. "Realtime Computer Vision with OpenCV". In: *Queue* (2012), pp. 40–56.

[70] Xukan Ran et al. "Deepdecision: A mobile deep learning framework for edge video analytics". In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE. 2018, pp. 1421–1429.

[71] *Real Time Protocol (RTP).* URL: `http://icapeople.epfl.ch/thiran/CoursED/RTP.pdf`.

[72] *Realtime transport protocol paraeters.* URL: `http://www.iana.org/assignments/rtp-parameters/rtp-parameters.xhtml`.

[73] Henri Rebecq et al. "High Speed and High Dynamic Range Video with an Event Camera". In: *arXiv preprint arXiv:1906.07165* (2019).

[74] Henri Rebecq et al. “High Speed and High Dynamic Range Video with an Event Camera”. In: *CoRR* abs/1906.07165 (2019). arXiv: 1906.07165. URL: http://arxiv.org/abs/1906.07165.

[75] *RTP: A Transport Protocol for Real-Time Applications*. URL: https://tools.ietf.org/html/rfc1889.

[76] S. Sarkar, S. Chatterjee, and S. Misra. “Assessment of the Suitability of Fog Computing in the Context of Internet of Things”. In: *IEEE Transactions on Cloud Computing* 6.1 (2015), pp. 46–59. ISSN: 2168-7161. DOI: 10.1109/TCC.2015.2485206.

[77] M. Satyanarayanan. “The Emergence of Edge Computing”. In: *Computer* 50.1 (2017), pp. 30–39. ISSN: 0018-9162. DOI: 10.1109/MC.2017.9.

[78] *Simple algorithme de détection de mouvement avec OpenCV JAVA*. URL: https://ratiler.wordpress.com/2014/09/08/detection-de-mouvement-avec-javacv/.

[79] M. Simsek et al. “5G-Enabled Tactile Internet”. In: *IEEE Journal on Selected Areas in Communications* 34.3 (2016), pp. 460–473. ISSN: 0733-8716. DOI: 10.1109/JSAC.2016.2525398.

[80] T. Taleb et al. “On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration”. In: *IEEE Communications Surveys Tutorials* 19.3 (2017), pp. 1657–1681. DOI: 10.1109/COMST.2017.2705720.

[81] David Tedaldi et al. “Feature detection and tracking with the dynamic and active-pixel vision sensor (DAVIS)”. In: *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. IEEE. 2016, pp. 1–7.

[82] *The reality of VR/AR growth*. URL: https://techcrunch.com/2017/01/11/the-reality-of-vrar-growth/?guccounter=1&guce_referrer_us=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_cs=Xs9Atn3eqXWQ2jIKKDITFg.

[83] *tshark*. URL: `https://www.wireshark.org/docs/man-pages/tshark.html`.

[84] *Tutorial*. URL: `https://inivation.com/support/software/jaer/h.aghj6wqgi7r4`.

[85] *Understanding UDP and TCP for live streaming*. URL: `https://www.oodlestechnologies.com/blogs/Why-UDP-is-preferred-for-Live-Streaming/`.

[86] *University of Zurich,Robotics and Perception Group*. URL: `https://rpg.ifi.uzh.ch`.

[87] "Virtual Reality and Augmented Reality Device Sales to Hit 11.9 Billion of dollars and 99 Million Devices in 2021". In: URL: `https://www.ccsinsight.com/press/company-news/3226-virtual-reality-and-augmented-reality-device-sales-to-hit-119-billion-and-99-million-devices-in-2021`.

[88] *VisualVM*. URL: `https://visualvm.github.io/`.

[89] *VisualVM features*. URL: `http://visualvm.github.io/features.html`.

[90] *VOT2015 Dataset*. URL: `http://www.votchallenge.net/vot2015/dataset.html`.

[91] *Why DVS*. URL: `https://inivation.com/dvs/`.

[92] Z. Xiang et al. "Reducing Latency in Virtual Machines: Enabling Tactile Internet for Human-Machine Co-Working". In: *IEEE Journal on Selected Areas in Communications* 37.5 (2019), pp. 1098–1116. ISSN: 0733-8716. DOI: `10.1109/JSAC.2019.2906788`.

[93] J. Xu et al. "Zenith: Utility-Aware Resource Allocation for Edge Computing". In: *Proc. of IEEE EDGE*. 2017, pp. 47–54. DOI: `10.1109/IEEE.EDGE.2017.15`.

[94] Wuyang Zhang et al. "Hetero-Edge: Orchestration of Real-time Vision Applications on Heterogeneous Edge Clouds". In: (Feb. 2019).

[95] Alex Zihao Zhu, Yibo Chen, and Kostas Daniilidis. “Realtime time synchronized event-based stereo”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 433–447.