Universidad Miguel Hernández de Elche

MÁSTER UNIVERSITARIO EN ROBÓTICA

ASAP: Adaptive Scheme for Asynchronous Processing of event-based vision algorithms

Trabajo de Fin de Máster
2019-2020

Autor: Raúl Tapia López
Tutor: Óscar Reinoso García
Cotutor: José Ramiro Martínez de Dios

Trabajo Fin de Máster
Máster en Robótica

ASAP: Adaptive Scheme for Asynchronous Processing of event-based vision algorithms

Autor:
Raúl Tapia López

Tutor:
Óscar Reinoso García
Catedrático de la Universidad Miguel Hernández de Elche

Cotutor:
José Ramiro Martínez de Dios
Catedrático de la Universidad de Sevilla

Universidad Miguel Hernández

Elche, 2020

A mis padres, por acompañarme en este camino que llega a su fin;
a mi hermana, a quien deseo suerte en el camino que le toca iniciar;
y a Sara, con quien recorreré todos los caminos que quedan por venir.

A dos referentes que hacen crecer mi pasión por la investigación cada día
y son piezas fundamentales en este proyecto:
Augusto, por su ayuda y enseñanzas;
y Ramiro, por su constante motivación y apoyo.

A los profesores del Máster Universitario en Robótica, por su acogida y entrega;
en especial a Óscar Reinoso, por su dedicación.

Abstract

Event cameras are neuromorphic sensors that capture changes in lighting intensity at pixel-level. They provide a number of advantages over conventional cameras and other perception sensors –such as high temporal resolution, low latency, high dynamic range or low power consumption– which have led to a revolution from a research and a commercial point of view. The potential applications of event cameras have motivated increasing interest in the robotics community.

The processing of events generated by the relative movement between the camera and the scene entails a paradigm shift in computer vision. Many algorithms in the literature group events to form fixed-frame images. However, these approaches do not fully exploit the advantages of the sensors. In order to cope with this issue, the development of asynchronous processing algorithms is required. This is a research effort that involves a review of all existing perception techniques.

One of the main problems of asynchronous processing is the management of the event stream. Some works feed the perception algorithms event-by-event, which provides low latency at a very high computational cost. Other approaches group the events in packets to reduce the computational cost, decreasing the responsiveness of the event-based algorithms.

This project presents ASAP (*Adaptive Scheme for Asynchronous Processing*), a solution based on a trade-off between event-by-event and event packet approaches. It is a framework whose aim is to feed event-based algorithms as soon as possible while avoiding the overflow caused by an excess of events in a short period of time.

The focus of this document is to describe the design, development and implementation of a novel method to achieve efficient processing while adapting to the environment –i.e., the generation of events– and the requirements of the perception algorithms through variable-sized event packets. Moreover, experimental results are presented to validate the correct operation of ASAP on-board a multirotor aerial vehicle.

Resumen

Las cámaras de eventos son sensores neuromórficos que capturan cambios en la intensidad luminosa a nivel de píxel. Proporcionan una serie de ventajas frente a las cámaras convencionales y otros sensores de percepción –alta resolución temporal, baja latencia, alto rango dinámico, bajo consumo de energía...– que han supuesto una revolución desde el punto de vista comercial y de investigación. Las potenciales aplicaciones de las cámaras de eventos han motivado un creciente interés en el ámbito de la robótica.

El procesamiento de los eventos generados por el movimiento relativo entre la cámara y la escena implica un cambio de paradigma en la visión artificial. Muchos algoritmos presentes en la literatura agrupan eventos para formar imágenes a una frecuencia constante, sin embargo, dichos enfoques no explotan al máximo las ventajas de estos sensores. Para afrontar este problema, se requiere el desarrollo de algoritmos de procesamiento asíncrono. Este es un esfuerzo de investigación que implica una revisión de todas las técnicas de percepción existentes.

Uno de los principales problemas del procesamiento asíncrono es la gestión del flujo de eventos. Algunos trabajos alimentan los algoritmos de percepción evento a evento, lo que proporciona una baja latencia con un coste computacional muy elevado. Otros enfoques agrupan los eventos en paquetes para reducir el coste computacional, disminuyendo la capacidad de respuesta de los algoritmos basados en eventos.

Este proyecto presenta ASAP (*Adaptive Scheme for Asynchronous Processing*), una solución basada en un compromiso entre ambos enfoques. Se trata de un marco de trabajo cuyo objetivo es alimentar a los algoritmos basados en eventos tan pronto como sea posible, evitando al mismo tiempo el desbordamiento causado por un exceso de eventos en un corto período de tiempo.

El objetivo de este documento es describir el diseño, desarrollo e implementación de un método novedoso para lograr un procesamiento eficiente a la vez que se produce una adaptación al entorno –es decir, a la generación de eventos– y a los requisitos de los algoritmos de percepción, utilizando para ello paquetes de eventos de tamaño variable. Además, se presentan resultados experimentales para validar el correcto funcionamiento de ASAP en un vehículo aéreo multirotor.

Short Contents

Contents

List of Figures

List of Tables

List of Algorithms

Notation

$\sim$	Approximately
$:=$	Assign, update
$\in$	Belongs to
Δ	Increment
$\bar{x}$	Normalized value of x
$\mathbf{A}^\top$	Transpose of matrix $\mathbf{A}$

1 Introduction

Event cameras are neuromorphic sensors capable of detecting changes in lighting intensity at pixel-level with microsecond temporal resolution and high dynamic range. The many advantages of this new type of sensor have attracted an increasing research interest in robotics in recent years, especially for real-time applications or variable lighting conditions scenarios.

Event processing involves a paradigm shift for computer vision. Although it is possible to group events to form images –i.e., event frames–, it does not exploit all the advantages of event cameras. Therefore, it will be necessary to redesign the traditional vision techniques to achieve event-based algorithms.

Most of the event-based algorithms in the literature are fed with event packets –i.e., grouped events–. This approach can cause a lack of responsiveness in cases with a low event generation rate. In contrast, other algorithms are fed event-by-event, which carries a risk of overflowing –especially with high event generation rate–.

1.1 Objective

The choice between event packing or event-by-event approach is a trade-off between responsiveness and risk of overflowing. This document presents an adaptive scheme for asynchronous event processing. The objective of the method is to manage the event stream using variable-size packets to optimize event processing. The aim is to feed event-based algorithms in such a way that an efficient and responsiveness behaviour is achieved and, at the same time, to prevent overflowing.

Low contrast or low dynamic scenes will generate few events, so small packets –even event-by-event– should be used to avoid inefficient processing. For a high rate of input events (e.g. aggressive manoeuvres on board a robot), the packets should be large in order to avoid non real-time processing. These ideas are the basis for the adaptive scheme that will be presented in this document.

1.2 Contribution

Currently in event processing related literature, two approaches are used in order to input the generated events into the vision algorithm: event-by-event and event packets. Both alternatives have advantages and disadvantages. Event-packing prevents algorithm computing saturation but it is not always efficient, while event-by-event approach is close real-time but can produce overflowing.

This work presents ASAP (*Adaptive Scheme for Asynchronous Processing*), a framework whose goal is to take advantage of both options using packets that vary in size according to the event-based algorithm computational requirements.

The project focuses on four main goals: the design of an efficient and innovative method for asynchronous processing, its development, its implementation on board an aerial robot and its experimental validation.

Achieving these goals is a major step in event-based perception towards a new form of communication between low-level event acquisition and event-based algorithms. ASAP is designed to be a modular friendly-use framework easy to integrate with the different event-based algorithms.

Previous work in this same direction has resulted in the publication of a paper in an ICRA 2020 workshop. Due to its innovation in the field of event processing, it is intended to extend the work done to publish in a journal.

1.3 Context

1.3.1 ASAP

This project is an extension of a previous work [73] published in IEEE International Conference on Robotics and Automation (ICRA) 2020 Workshop on Unconventional Sensors in Robotics. A first prototype of ASAP was presented there to manage the event stream that feed an event-based clustering algorithm. The experimental results showed that an efficient algorithm processing can be achieved.

1.3.2 H2020 GRIFFIN

This work is part of the GRIFFIN ERC Advanced Grant 2017 (788247) [74] [75] european project, granted to the GRVC Robotics Laboratory [35] of the University of Seville. The aim of GRIFFIN (*General compliant aerial Robotic manipulation system Integrating Fixed and Flapping wings to INcrease range and safety*) is to create a unified framework for the development of autonomous flapping-wing robots –also called ornithopters– with dexterous manipulation capabilities.

GRIFFIN ornithopters (Figure 1.1) will be able to fly minimizing energy consumption and to perform manipulation tasks while held on a surface. In many scenarios, this

kind of robot allows safer operation than rotorcrafts, especially in human interaction tasks.

Figure 1.1 GRIFFIN ornithopter prototypes designs by GRVC Robotics Laboratory of the University of Seville.

1.3.3 Visual Perception for Winged Aerial Robots

Visual perception has been widely used in aerial robotics due to the size, weight, and consumption of the sensors. Flapping wing robots generate a challenge for visual perception because fast-flying manoeuvrers cause high vibrations, which produce motion blur in the images. For this reason, event sensors are seen as a desirable option to place on board an ornithopter.

Event-based vision not only prevents motion blur, but also makes working under strong lighting changes possible. To these advantages, it must be added the lower consumption and lower latency compared to conventional sensors.

Gómez et al. [28] presented a paper showing a comparison between a monocular camera and an event camera on board the platform in Figure 1.2. The authors concluded that considering the nature of the orthopter's flight –high speed manoeuvrers and vibrations–, event-based cameras have better properties than frame-based cameras.

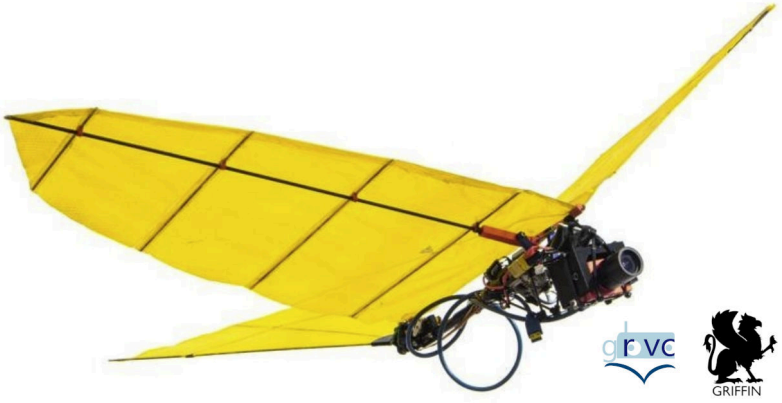

Figure 1.2 GRIFFIN ornithopter prototype with an onboard DAVIS346 event camera. Adapted from [28].

1.3.4 ASAP Code Developing

All the code used in this project has been implemented in C++ 11 under ROS Melodic Morenia framework. ROS (Robot Operating System) is a middleware for robotics software development. It provides services for heterogeneous computer cluster (e.g., message communication, client-server structures, hardware abstraction, low-level device control...). Its advantages include modular and reusable code, concurrency, integration of developing tools –debugging, visualization, simulation...— and a wide use in robotics.

1.3.5 DAVIS346 Event Camera

A DAVIS346 event camera (Figure 1.3) was used during design and for experimental testing. This is a 346 x 260 pixels DVS (Dynamic Vision Sensor) event camera with a 1 μs temporal resolution, over 20 μs latency, 120 dB dynamic range. DAVIS sensors include an APS (Active Pixel Sensor), so it also outputs fixed frame-rate grayscale images.

Figure 1.3 DAVIS346 Event Camera.

1.4 Structure

This document consists of 9 chapters and 1 appendix. Chapter 2 presents the state of the art of event processing. Chapter 3 provides an overview of the ASAP software. Chapters 4 and 5 detail the modules that compose the developed framework. Chapter 6 presents the experiments performed in order to validate the proposed method. In Chapter 7, an analysis of the obtained results is presented.

Finally, in Chapter 8 the conclusions drawn during the realization of this project and its possible future work are presented. Chapter 9 contains a Spanish translation of these conclusions. Appendix A present the ASAP algorithm for implementation.

2 State of the Art

2.1 Introduction

This chapter will present the state of the art of bio-inspired event-based computer vision. First, an introduction to event camera sensors –principle of operation, different designs, advantages over standard camera sensors, commercial devices...– will be given. This will be followed by the state of the literature related to event processing.

2.2 Event Cameras

An event camera is a bio-inspired neuromorphic sensor whose operation differs from that of a conventional camera. A standard camera acquires images at a fixed frame rate, while an event camera captures light intensity changes asynchronously. Therefore, event cameras will only output information in dynamic environments with contrast –something in the scene moves or the camera moves–.

This kind of performance provides many advantages over traditional image acquisition, although it changes the paradigm of existing perception algorithms. Even though event cameras are commercially available since 2008 [38], the literature related to this research field [22] and a mass production interest by companies (e.g., iniVation [29], Prophesee [57] or Samsung [77]) has grown in a short time. This highlights the many advantages and the commercial interest in exploiting these new sensors.

2.2.1 Principle of Operation

A digital camera is an electronic device that captures images using a sensor capable of detecting the intensity with which the photons collide on it (Figure 2.1). This kind of sensors is composed of an array of light-sensitive elements. Each element is known as pixel and the number of pixels in the array defines the sensor resolution. Currently, the most commonly used sensors in digital cameras are CCD (Charged Coupled Devices) and CMOS (Complementary Metal-Oxide

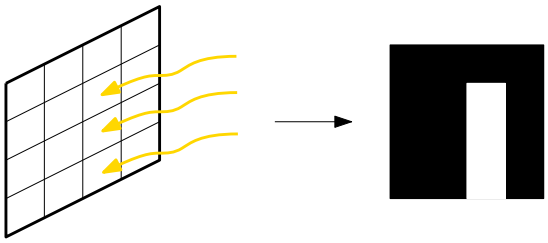

Figure 2.1 Standard camera operation.

Semiconductor).

In contrast to conventional cameras, event cameras capture light intensity changes –called events– asynchronously and independently –i.e., at pixel-level–. When a pixel experiences a change to a higher intensity level, the camera will take it as a positive event, while if a change to a lower level occurs, it will be interpreted as a negative event (Figure 2.2). This positive/negative state is called event polarity.

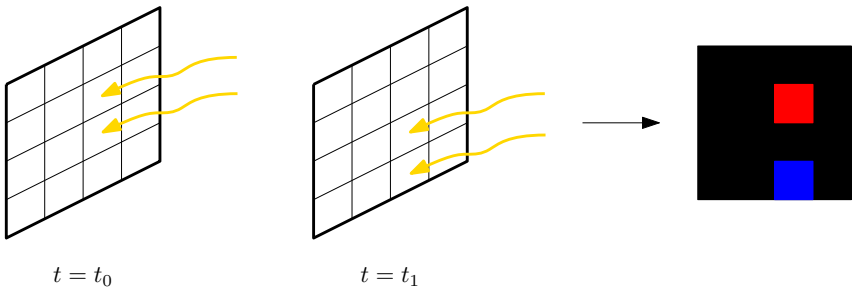

Figure 2.2 Event camera operation.

Figures 2.3, 2.4 and 2.5 are intended to be an event camera behaviour demonstration. The experiment –based on [38]– consisted of a wheel spinning (Figure 2.3). When the rotation speed is constant, the events pattern repeats over time (Figure 2.4). On the other hand, with variable rotation speed, the helicoid pitch is lower the higher the speed (Figure 2.5).

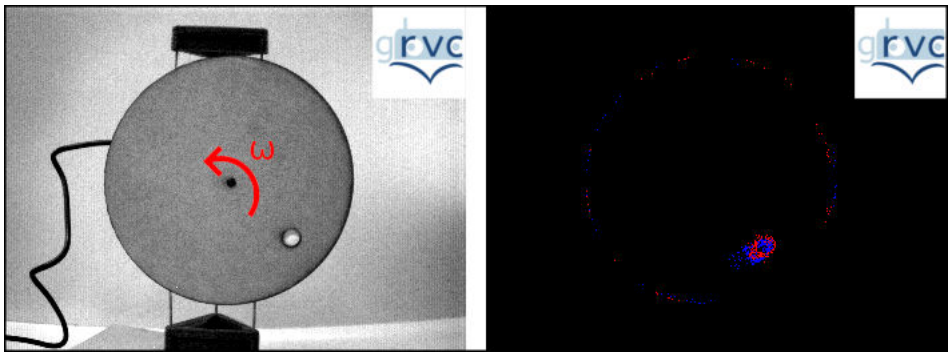

Figure 2.3 Standard image (left) and event frame image (right).

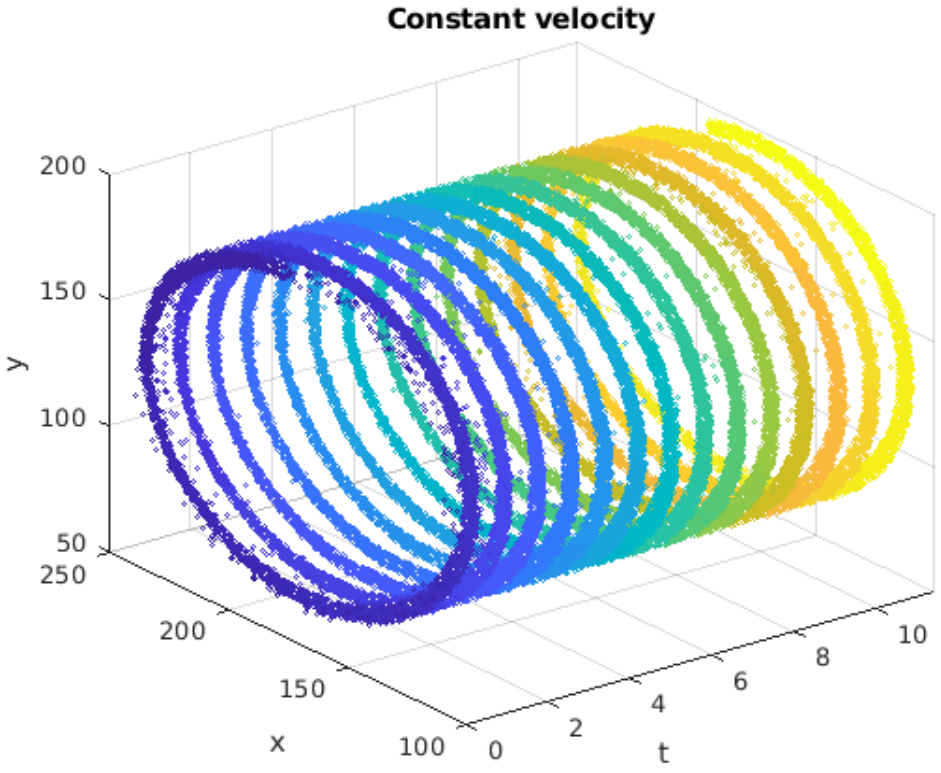

Figure 2.4 Generated events with constant angular velocity.

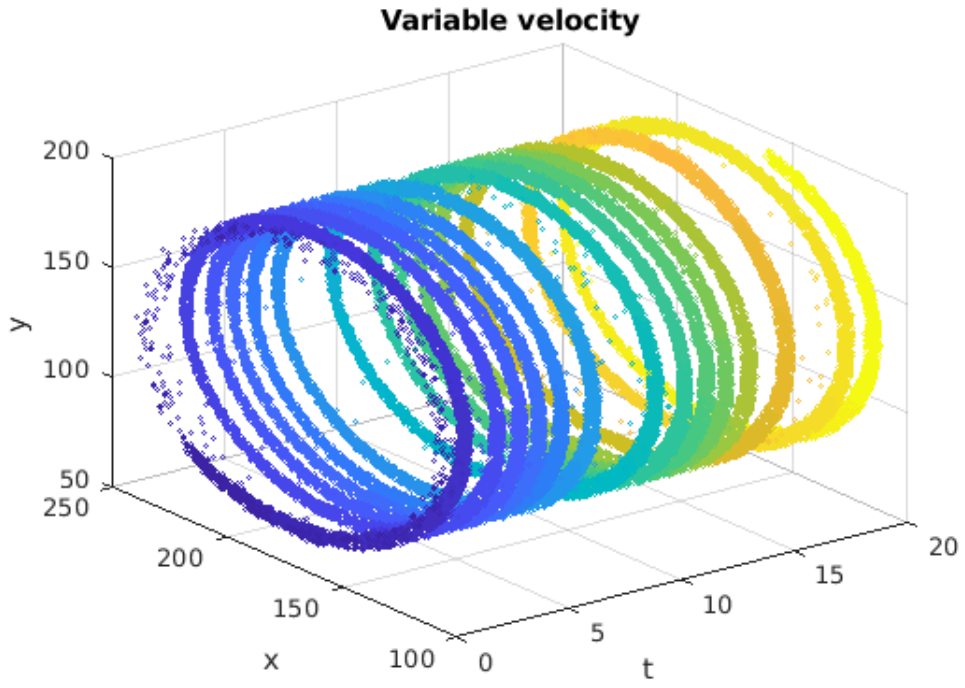

Figure 2.5 Generated events with variable angular velocity.

2.2.2 Event Camera Designs

The first approach to bio-inspired vision sensors –so-called silicon retinas [20] [53] [54]– was the device developed by M. Mahowald in his thesis [41]. Its many limitations (e.g., very large pixels, bias circuit for each node which must be adjusted, non-homogeneous pixel response...) made it not a system for a practical use. However, different designs over the years have overcome these problems to achieve fully functional devices [56].

The following sections will present the most commonly used event-based vision sensors [83]: the Dynamic Vision Sensor (DVS) [38], the Asynchronous Time-based Image Sensor (ATIS) [55] and the Dynamic and Active Pixel Vision Sensor (DAVIS) [11]. There are other less widely used retina designs [40], such as the asynchronous Parvo–Magno retina model [79] [80], the octopus retina [19] or the spatial contrast and orientation vision sensor (VISe) [66].

Dynamic Vision Sensor (DVS)

The DVS [38] is based on previous frame-based silicon retina designs [21] [33] [39] which used a photoreceptor capacitively coupled to a readout circuit. However, DVS quantizes relative intensity changes at pixel-level in continuous time –i.e., non frame-based–. Bio-inspirated and neuromorphic systems literature [20] [56] [40] highlights the advantages of this type of operation.

DVS pixels get light changes using a logarithmic photoreceptor circuit, a differencing circuit –for amplifying the changes– and two transistor comparators (Figure 2.6). Figure 2.7 illustrates the DVS pixel operation principle, where it is shown how DC mismatch between pixels can be solved by setting the differencing circuit output to a reset level after each event generation.

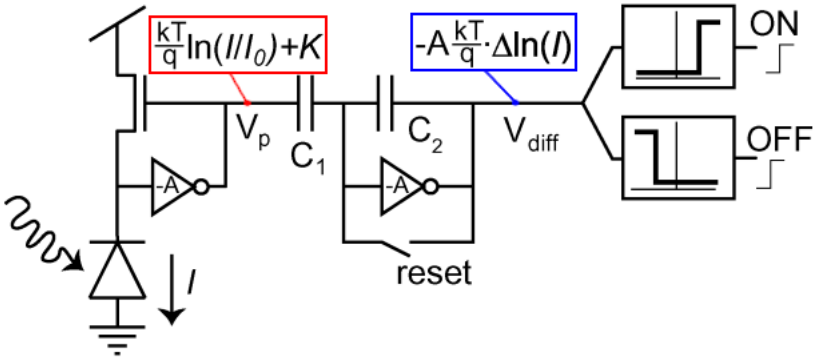

Figure 2.6 Simplified pixel schematic. Adapted from [38].

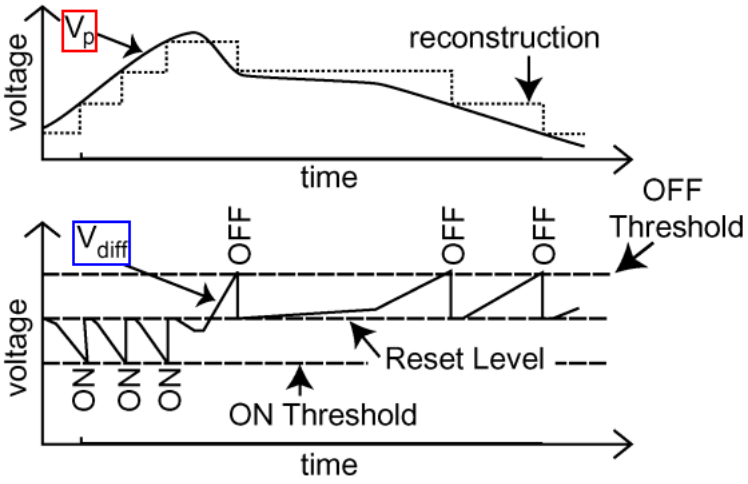

Figure 2.7 Pixel principle of operation. Adapted from [38].

Asynchronous Time-based Image Sensor (ATIS)

ATIS [55] is presented as a device which aims to combine different bio-inspired approaches in order to acquire asynchronous event-based information. The sensor incorporates an array of asynchronous pixels and pulse-width-modulation exposure measurement circuits.

Each pixel in ATIS is made of a DVS subpixel which activates another subpixel to read out the absolute intensity and which resets the differencing voltage level. Although an ATIS pixel allows intensity values to be obtained, its area is more than twice the DVS pixel area.

Dynamic and Active Pixel Vision Sensor (DAVIS)

The DAVIS [11] combines a DVS pixel and a active pixel sensor (APS) in the same pixel. This device allows to obtain intensity level with a much smaller pixel size than ATIS, since the photoreceptor is shared. The APC outputs information at a constant frame rate and has the standard camera limitations.

Figure 2.8 presents the DAVIS pixel schematic, where it can be seen that the photoreceptor circuit voltage serves as APS input.

Sensors Specifications Comparison

Table 2.1 aims to be a summary of the specifications of the three types of sensors above-mentioned. All the information has been extracted from [83], [40], [38], [55] and [11].

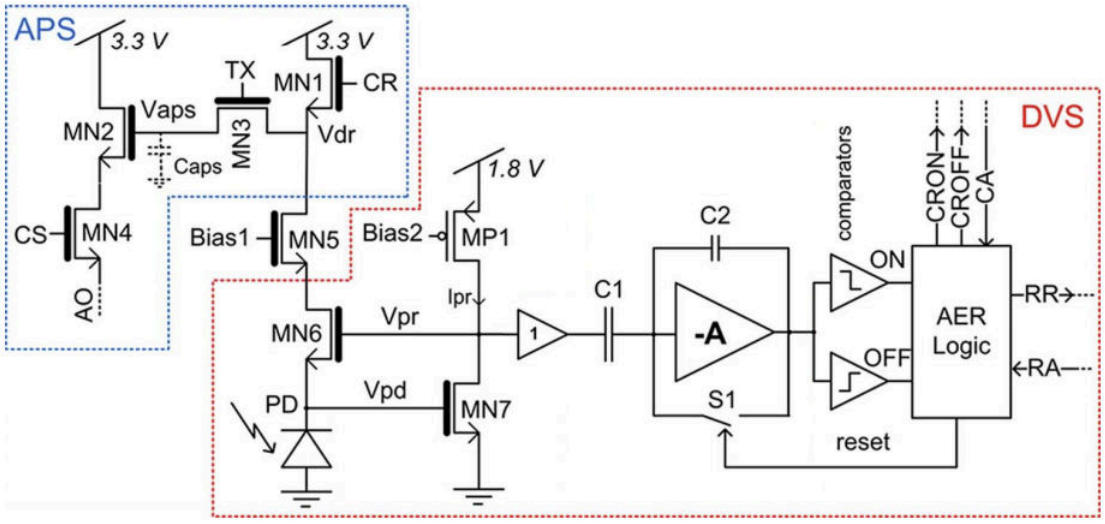

Figure 2.8 DAVIS pixel schematic. Adapted from [11].

Table 2.1 DVS, ATIS and DAVIS specifications.

	DVS	ATIS	DAVIS
Release	2006	2008	2013
Pixel size	40 $\mu m \times$ 40 μm	30 $\mu m \times$ 30 μm	18.5 $\mu m \times$ 18.5 μm
Sensor size	128 $px \times$ 128 px	304 $px \times$ 240 px	240 $px \times$ 180 px
Latency	15 μs	4 μs	3 μs
Dynamic range	120 dB	125 dB	130 dB
Consumption	24 mW	50 - 175 mW	5- 14 mW

2.2.3 Event Cameras Advantages

As stated above, the silicon retinas operation has a large number of advantages over standard cameras. This section will detail the most remarkable and useful advantages from the point of view of computer vision processing algorithms.

Low Latency

It is not necessary to wait for a global exposure time for each frame, but pixels are independent. This asynchronous behaviour makes a very low latency value possible.

High Temporal Resolution

Events are detected with a microsecond temporal resolution. Unlike traditional cameras, these sensors are able to capture very fast movements with no motion blur (Figure 2.9).

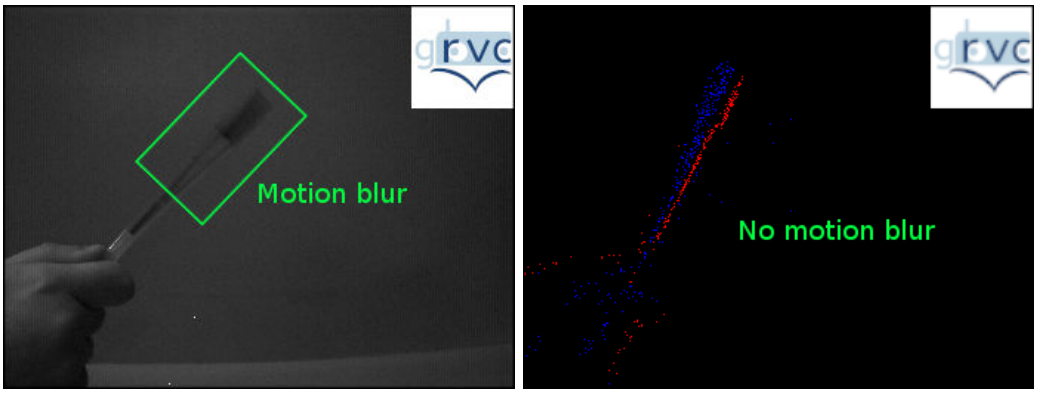

Figure 2.9 Motion blur in standard image (left) and event frame image (right).

High Dynamic Range

Event cameras have a high dynamic range (HDR), over 120 dB –standard cameras dynamic range is about 60 dB–. For that reason, silicon retinas can acquire information under different lighting conditions (Figure 2.10). Like a human eye, these sensors are capable of adapting to scenarios from very dark to very bright.

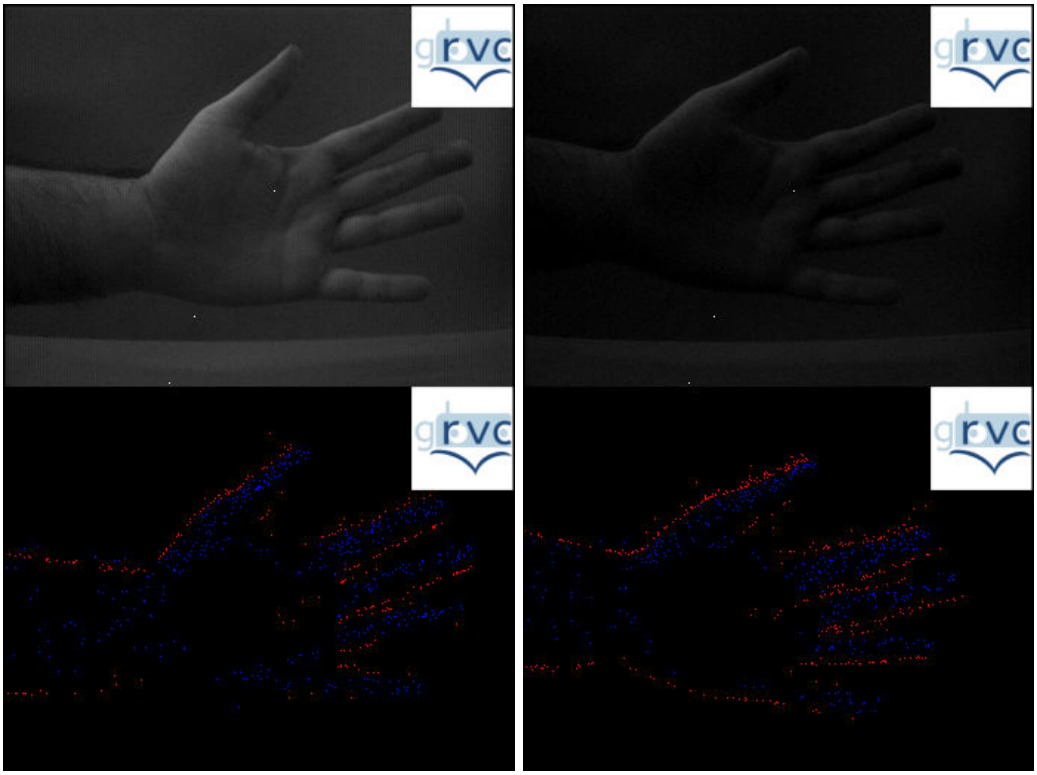

Figure 2.10 Standard image (up) and event frame image (down) for two different lighting condition.

Low Power Consumption

Artificial retinas have a power consumption about tens of milliwatt. This low power consumption is mainly due to the fact that redundant information is avoided –energy is only used on pixels whose intensity changes–.

2.2.4 Applications

Event-based processing has brought about a paradigm shift in computer vision algorithms. Traditional image-based algorithms must be redesigned to work with asynchronous processing in order to take full advantage of the event cameras. This new paradigm is very challenging from a research point of view.

This section will describe some of the many algorithms present in the literature in order to give an idea of the many applications of these sensors. Generally, these algorithms focus on scenarios where real-time response, low power and uncontrolled lighting conditions operation are needed.

Feature Detection

An event by itself does not provide information about its neighbourhood, so it is impossible to know if it belongs to a corner. Clady et al. [18] propose as a solution to accumulate the events to create a map of the timestamp of the latest event for each pixel, called SAE (Surface of Active Events). They estimate corners by fitting planes to the SAE and searching for intersections.

Vasco et al. [76] present a faster method –plane fitting is a costly operation, so a large number of events per second cannot be processed–. They use two binary SAEs on which they apply the Harris corner detector algorithm. Figure 2.11 shows some experimental results.

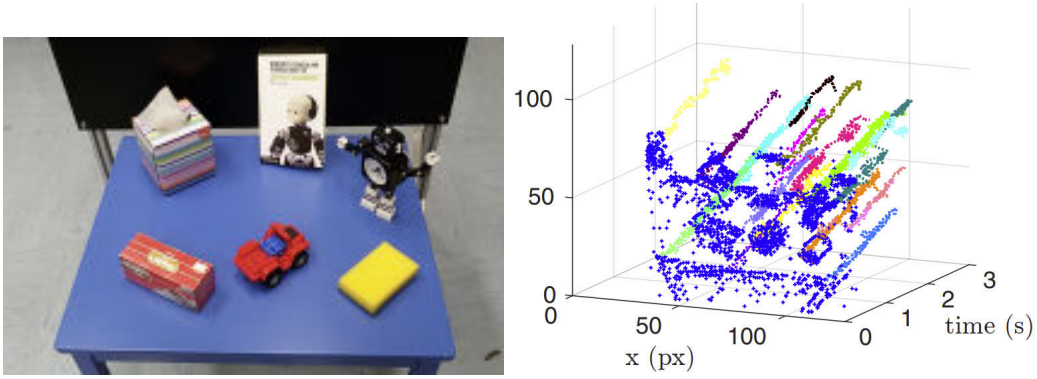

Figure 2.11 Experimental setup (left) and detected corners over time (right). Adapted from [76].

However, event-based Harris corner detection is not computationally efficient due to the required matrix operations. Mueggler et al. [44] propose an event-based detector relied on FAST corner detector. Their algorithm compares each event

with the SAE events arranged in a circumference around that pixel (Figure 2.12). Alzugaray et al. [3] also based on FAST for their detector design, but with an arc of circumference searching (Figure 2.13).

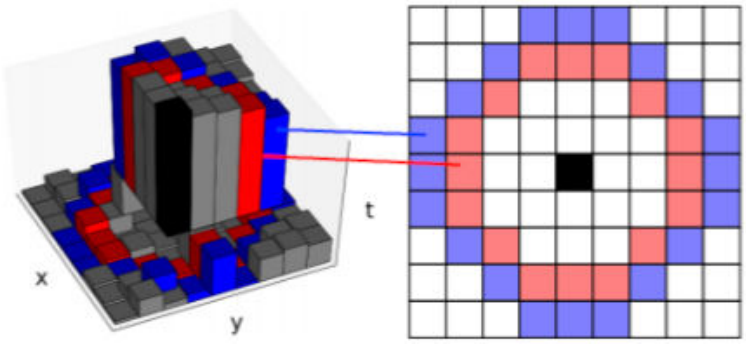

Figure 2.12 Surface of active events representation. Adapted from [44].

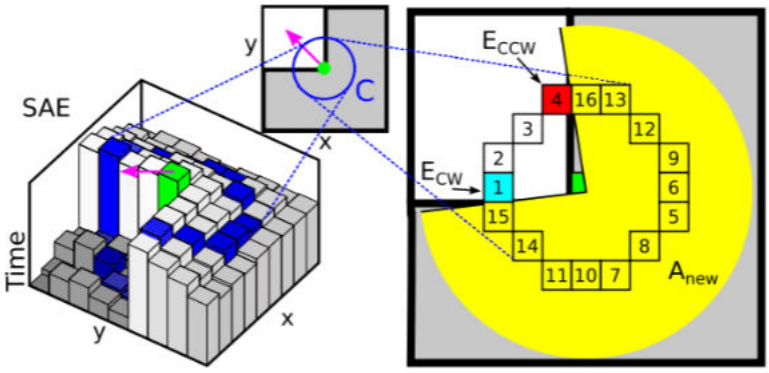

Figure 2.13 Surface of active events representation. Adapted from [3].

An improved Harris-based corner detector is presented by Li et al. in [37]. They implemented a Global Surface of Active Event (G-SAE) for enhancing real-time performance, reaching eight times the speed of Vasco et al. [76] detector.

Tracking

Zhu et al. [81] and Alzugaray et al. [2] are two examples of feature tracking. The first publication uses a data association modelled with probabilities computed from optical flow. The second one is presented as an asynchronous framework to track event features using an underlying directed graph representation.

Pose tracking algorithm from Mueggler et al. [47] was used on board a multirotor that was aggressively manoeuvred, taking advantage of the fact that event cameras are not affected by motion blur.

Rodríguez-Gómez et al. presented in [63] an asynchronous event-based tracking for intrusion monitoring under different lighting conditions (Figure 2.14). Their

method employs an event clustering and a feature tracking modules optimized to support event-based processing computational constraints.

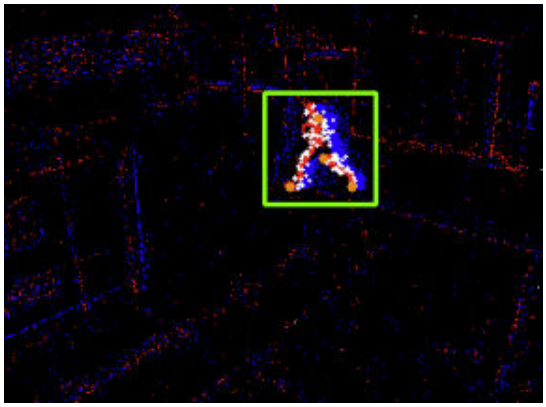

Figure 2.14 Intruder monitoring during night. Adapted from [63].

Segmentation and Clustering

Glover et al. [25] [26] propose a method to detect and track a moving ball using a Hough transform approach (Figure 2.15). The paper extends the Hough-based circle detection algorithm using optical flow, extracted from spatio-temporal event space.

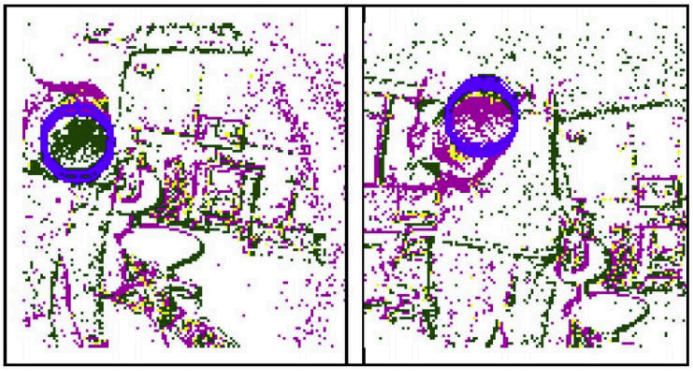

Figure 2.15 Example of ball detection. Adapted from [25].

A contour detector is presented by Barranco et al. in [7]. Authors compute contours and scene segmentation (Figure 2.16) using a Structures Random Forest (SRF) on event-based features for encoding information to feed a classifier.

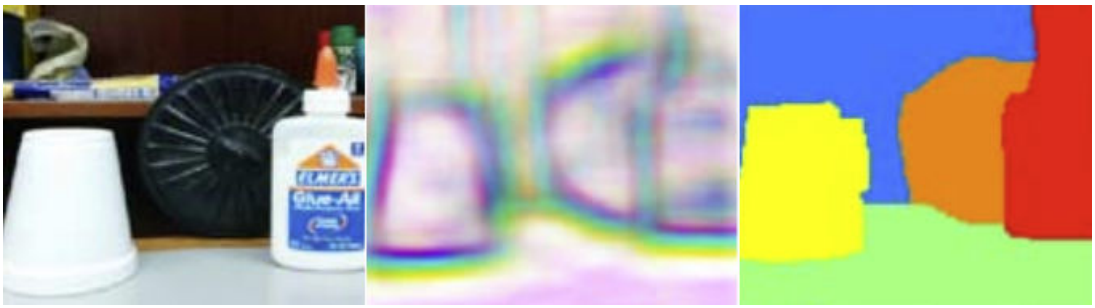

Figure 2.16 Real scene (left), predicted contours (center) and predicted ownership (right). Adapted from [7].

The method proposed by Barranco et al. [6] is an example on event-based clustering algorithm (Figure 2.17) relying on probability density function to represent the feature space. Another instance is the aforementioned publication [63] by Rodríguez-Gómez et al., who use a real-time asynchronous clustering technique to group events created by an intruder in the scene.

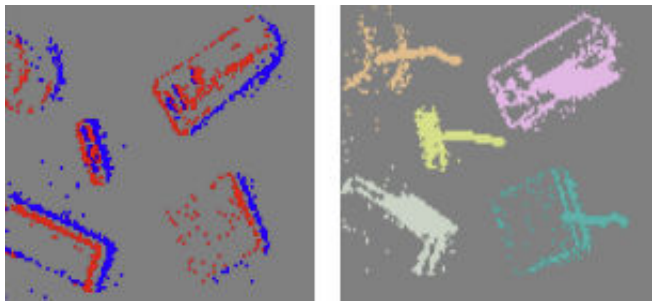

Figure 2.17 Image of events (left) and clustering output (right). Extracted from [6].

Alonso et al. [1] designed a semantic segmentation convolutional neural network fed only with events. Semantic labels are generated from grayscale image.

Object Recognition

Employing event-based data with machine learning tools –such as neural networks– allows object recognition. Serrano-Gotarredona et al. [69] –who use a Convolutional Neural Network–, Moeys et al. [43] –also using a CNN–, Li et al. [36] –with a random forest based classifier– or Ghosh et al. [24] –based on an Extreme Learning Machine (ELM)– are examples of recognition applications.

Optical Flow

Publication instances about event-based visual motion estimation are: Benosman et al. [9] –where results demonstrate optical flow computation current limitations can be overcome by using event-based acquisition, since high temporal resolution allows optical flow with microsecond accuracy–, Orchard et al. [51] –whose authors describe a visual motion estimation which uses a spiking neural network to exploit

high temporal resolution event data– or Barranco et al. [5] –where optical flow computation (Figure 2.18) is used for a contour motion estimation–.

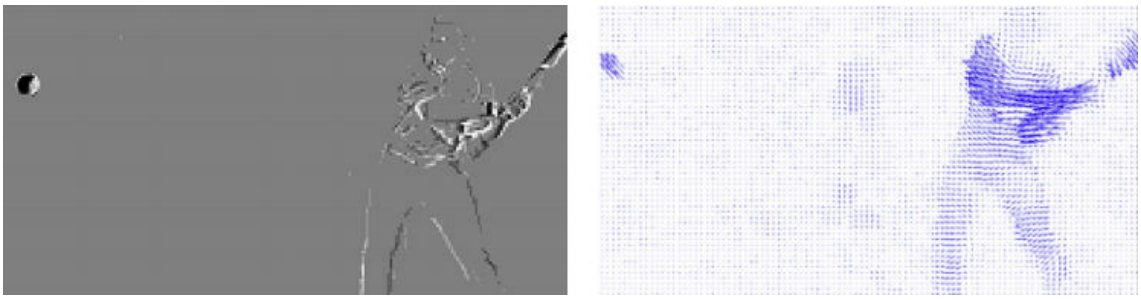

Figure 2.18 Events image (left) and optical flow (right). Adapted from [5].

3D Reconstruction

An event-based multi-view reconstruction is presented by Rebecq et al. in [60] and later extended in [59], where authors estimate 3D structure from an event camera with known trajectory using monocular depth estimation.

Also three-dimensional reconstruction using structured light is possible. Brandli et al. [12] employ a pulsed laser line extraction for terrain reconstruction. In the field of stereo depth estimation (Figure 2.19), there are papers such as Rogister et al. [64], Carneiro et al. [15] or Camuñas-Mesa et al. [13] [14].

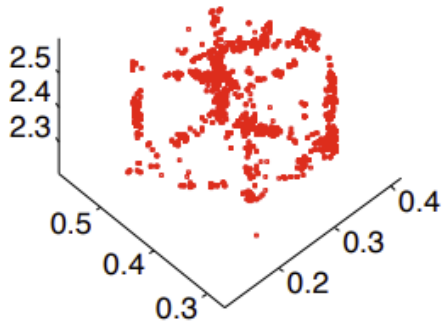

Figure 2.19 Event-based 3D reconstruction of a cube. Adapted from [15].

Visual Odometry and SLAM

Kueng et al. [34], Zhu et al. [82] and Censi et al. [17] are examples of event-based visual odometry algorithms. Both methods exploit the event processing advantages for low-latency response and high temporal resolution.

Mueggler et al. [46] propose a localization algorithm adapted to event cameras asynchronous operation. Gallego et al. [23] describe another localization method which uses the contrast residual as a measure of the pose estimation accuracy.

Event-based visual SLAM (Simultaneous Localization And Mapping) (Figure 2.20) has also been developed in recent years, with publications such as Weikersdorfer et al. [78], Kim et al. [32], Rebecq et al. [61] or Rosinol et al. [65], among others.

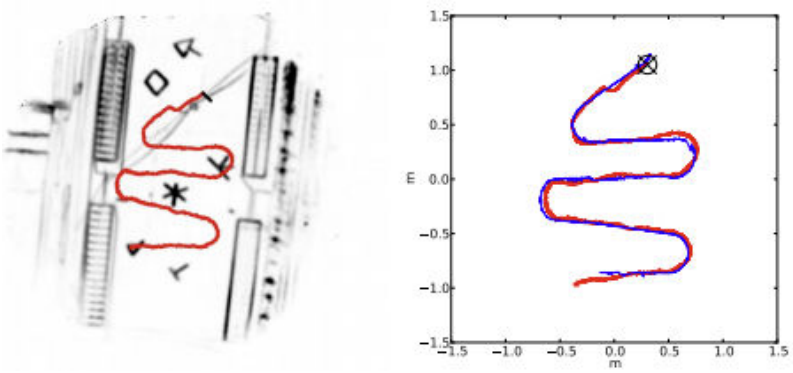

Figure 2.20 Map and path generated (left) and trajectory (right). Adapted from [78].

Visual Servoing

The field of event-based visual servoing is a currently growing and challenging line of research where there is still much to be done. It is worth mentioning the recent publication by Muthusamy et al. [48], who has developed an eye-in-hand visual servoing for a manipulator robot (Figure 2.21).

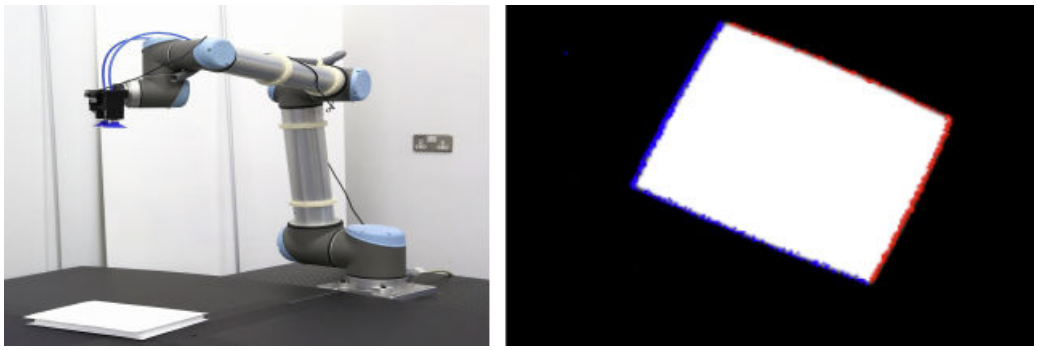

Figure 2.21 Manipulator robot for visual servoing (left) and image of events (right). Adapted from [48].

Standard Images Reconstruction

Is it possible to reconstruct grayscale images from events? Some works such as Barua et al. [8], Kim et al. [31], Scheerlinck et al. [68] or Reinbacher et al. [62] have shown that, using machine learning algorithms –e.g., manifold regularization– or optimization techniques, good accuracy results (Figure 2.22) are obtained.

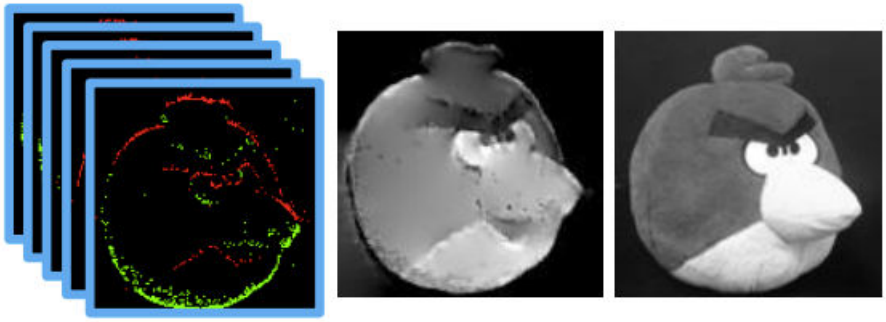

Figure 2.22 Images of events (left), intensity image reconstruction (center) and intensity image reference (right). Adapted from [8].

2.2.5 Commercial Devices

Different advantages and applications have made some companies interested in producing event cameras. These commercial products facilitate in many cases the sensor integration in robots for research purposes in the field of event-based computer vision. Next, some of the models available on the market are shown.

Prophesee

Prophesee [57] has developed ATIS-Gen1, CD-Gen2, ATIS-Gen3, CD-Gen3, ATIS-Gen4 and ATIS-Gen4. Currently, the company markets an event camera Evaluation Kit (EVK). Their last release is the IMAGO VisionCam –powered by Prophesee– (Figure 2.23), announced as the first industrial event-based vision system.

Figure 2.23 IMAGO VisionCam, powered by Prophesee. Extracted from [57].

iniVation

iniVation [29] has four discontinued products: eDVS, DVS128, DVS240 and DAVIS240 (Figure 2.24). At the moment, they market the new models DAVIS346, DVXplorer and DVXplorer Lite (Figure 2.25).

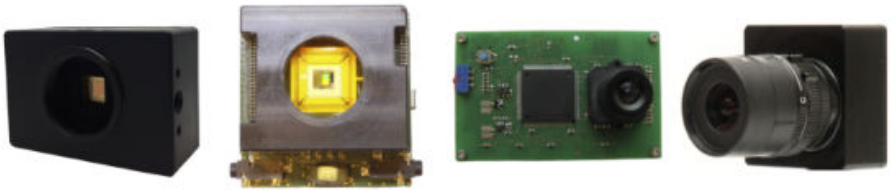

Figure 2.24 Left to right. DVS240, DAVIS240, eDVS, DVS128. Adapted from [29].

Figure 2.25 Left to right. DAVIS345, DVXplorer and DVXplorer Lite. Adapted from [29].

Samsung

Samsung Electronics [67] has also shown interest in this type of technology, releasing the DVS-Gen1, DVS-Gen2, DVS-Gen3 and DVS-Gen4 devices. Recently, this company has put up for sale the SmartThings Vision [77] (Figure 2.26), a commercial product for home monitoring.

Figure 2.26 Samsung SmartThings Vision. Extracted from [77].

Insightness

Insightness [30] has developed the Rino 3 Silicon Eye (Figure 2.27). Among its many uses are UAS collision avoidance or augmented reality applications.

CelePixel

CelePixel [16] released in 2018 the first one megapixel event-camera sensor, called CeleX-V (1280x800). This device is the result of different iteration over

Figure 2.27 Insightness Rino 3 Silicon Eye. Extracted from [30].

the past few years: CeleX-I (64x64) in 2012, CeleX-II (192x160) and CeleX-III (384x320) in 2015 and CeleX-IV (768x640) in 2017.

Commercial Devices Comparative

Table 2.2 shows a comparison between the specifications of the above-mentioned technologies. All values has been extracted from [57], [29], [67], [52], [71], [30], [16] and [22].

2.3 Event Processing

Artificial retinas asynchronous behaviour means that these sensors output is not constant frame rate images, but a timestamped events stream. This kind of read access is called Address-Event Representation (AER) [40] [10]. This section will describe the main characteristics of AER and its advantages over frame-based representation.

2.3.1 AER Vision Sensor

Address-Event Representation is bio-inspired by the neurons communication. Pixels asynchronously output event position –i.e., their address– (Figure 2.28) when they detect a significant intensity change.

Power consumption required for this type of operation is much lower compared to frame-based sensors. In addition, a much higher temporal resolution can be achieved, since it is not necessary to wait for an global exposure time –each pixel output is independent–.

Table 2.2 Comparative of commercial event-based technologies.

		Release	Resolution px^2	Pixel size μm^2	Sensor size mm^2	Latency μs	Dyn. range dB	Power cons. mW
Prophesee	CD-Gen3	2017	640 × 480	15 × 15	9.6 × 7.2	12	120	36 - 95
	ATIS-Gen3	2017	480 × 360	20 × 20	9.6 × 7.2	12	120	25 - 87
	CD-Gen4	2020	1280 × 720	4.86 × 4.86	6.22 × 3.5	12	124	32 - 73
iniVation	eDVS	2008	128 × 128	40 × 40	6 × 6.6	12	120	23
	DVS128	2008	128 × 128	40 × 40	6.3 × 6	12	120	23
	DVS240	2014	240 × 180	18.5 × 18.5	5 × 5	12	120	5 - 14
	DAVIS240	2014	240 × 180	18.5 × 18.5	5 × 5	12	120	5 - 14
	DAVIS346	2017	346 × 260	18.5 × 18.5	8 × 6	20	120	10 - 170
	DVXplorer	2020	640 × 480	-	-	100	110	-
	DVXplorer Lite	2020	320 × 240	-	-	100	110	-
Samsung	DVS-Gen1	2014	640 × 480	9 × 9	9.7 × 8	-	66	15
	DVS-Gen2	2016	640 × 480	9 × 9	8 × 5.8	65 - 410	90	27 - 50
	DVS-Gen3	2018	640 × 480	9 × 9	8 × 5.8	50	90	40
	DVS-Gen4	2020	1280 × 960	4.95 × 4.95	8.4 × 7.6	150	100	130
Insightness	Rino 3	2018	320 × 262	13 × 13	5.3 × 5.3	125	100	20 - 70
CelePixel	CeleX-IV	2017	768 × 640	18 × 18	15.5 × 15.8	10	90	-
	CeleX-V	2019	1280 × 800	9.8 × 9.8	14.3 × 11.6	8	120	400

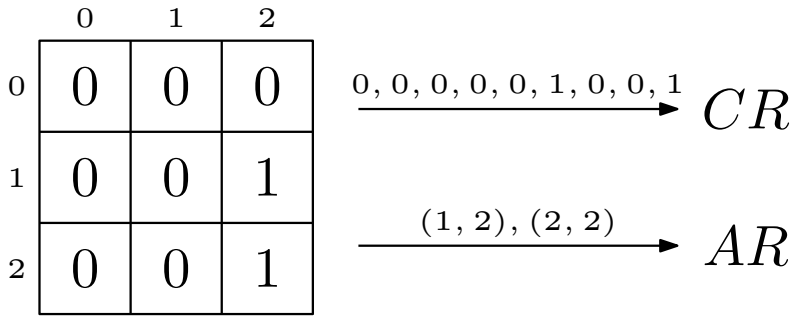

Figure 2.28 Address representation (AR) vs. content representation (CR).

According to Delbruck et al. [20], AER sensors can be classified into:

- Spatial Contrast (SC) sensors –which reduce spatial redundancy based on intensity ratios– and Spatial Difference (SD) sensors –which use intensity differences–.
- Temporal Contrast (TC) sensors –which reduce temporal redundancy based on relative intensity changes– and Temporal Difference (TD) sensors –which use absolute intensity changes–.
- Frame event sensors –which use a synchronous global exposure time– and asynchronous event sensor –with no global exposure–.

The vast majority of event sensors are asynchronous, so AER-based and asynchronous event-based concepts are usually used synonymously. Ruedi et al. [66] is an example of fixed frame –i.e. non-asynchronous– AER vision sensor.

In addition, AER is not limited to event-based sensors. There are a few standard intensity-value sensors such as Shoushun et al. [70], Culurciello et al. [19] or Azadmehr et al. [4]. However, these designs are expensive and do not provide significant latency reduction, so they are not very common.

2.3.2 Frame-Based vs. Event-based Algorithms

Algorithms fed with the event stream can group them to form frames –i.e. event images–. However, this kind of operation does not exploit all the advantages of silicon retinas and is computationally less efficient than event-based methods.

Rodríguez-Gómez [63] is an example of event-based processing. Authors show the advantages of asynchronous processing, especially useful for real-time applications which require high computational efficiency.

2.3.3 Single Events vs. Event Packets

In order to feed event-based algorithms, two main methods are employed: single events –events are sent and processed one by one– and event packets –events are packed before being sent to the processing algorithm–.

Event packeting (e.g., [27], [45] or [47]) prevents algorithm overflow but it is not

an efficient operation when algorithm can process events faster that they are received. On the other hand, single-events option (e.g., [42] or [63]) is the closest operation to real-time thanks to the event camera microsecond temporal resolution. However, this can result in an algorithm overflow, especially when many events are generated in a short period of time.

The most common option in literature is event packeting. Two approaches are used to group: packed by number of events –all packets are the same size– and packed by time –packets are sent periodically–. The greater the relative movement between scene and camera, the greater the number of events generated. Thus, a greater number of packets –if number packing is used– or larger packets –if time packing is used– will be generated.

2.4 Conclusions

This chapter has intended to give an idea of the revolution that event cameras represent for computer perception. These new sensors represent a very challenging paradigm shift for researchers that is bringing about a lot of possibilities.

In the event processing context, it can be stated that event-based algorithms are generally more efficient than frame-based algorithms because they allow all the advantages of silicon retinas to be exploited.

The choice between single-event based or event-packet based processing involve a compromise between responsiveness and overflow risk. In order to exploit the full capability of event-based vision, the project described in this document aims to design an adaptive scheme for events processing. A previous work [73] has already confirmed the possibility of optimizing the event packet size to obtain an efficient real-time behaviour without risk of overflowing.

3 ASAP: Adaptive Scheme for Asynchronous Processing

This chapter will present the scheme and general operation of ASAP framework. The objective of ASAP (*Adaptive Scheme for Asynchronous Processing*) is to modify in real-time the size of the event packets that will be input into the event-based vision algorithm according to its computational needs.

Afterwards, the operation of the two modules that compose the scheme –filtering module and packing module– will be introduced. The first module is used to reduce the computational consumption by removing some random events –avoiding redundant information–. The second one is designed to pack the events in a computationally efficient way.

Furthermore, the communication structure between the different ASAP components, the messages types that use these communications and the connection of the event-based algorithms will be explained.

3.1 ASAP Framework

The number of events generated will depend on the contrast and dynamics of the scene. To maintain a near real-time behaviour, events must be processed right after they are generated by the sensor. However, a large number of events in a small period of time could saturate the algorithm and cause overflowing. In these cases it is desirable to send them by groups.

Based on this knowledge, ASAP (Figure 3.1) aims to adapt the size of event packets to achieve the most efficient processing result. To this end, events will be sent to the algorithm *as soon as possible*, ensuring that there is no risk of saturation.

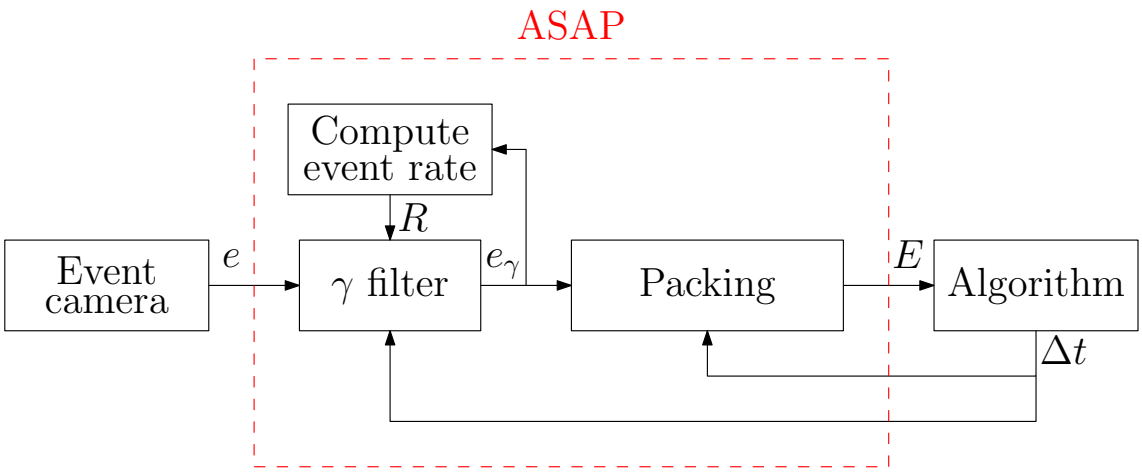

Figure 3.1 ASAP simplified scheme.

The modules presented in Figure 3.1 are as follows:

- **Event camera**: Events e are output from the vision sensor and fed into ASAP.
- γ **filter**: This preprocessing step is used to reduce the number of events to be packed, resulting in a lower e_γ rate. The reduction will be greater the higher the event rate and the higher the algorithm computational cost Δt.
- **Event rate computation**: Event rate R –i.e., events per second– is computed to suit the filtering.
- **Packing**: The size of the event packets E will be adjusted according to the algorithm temporal cost Δt.
- **Algorithm**: The vision event-based algorithm processes the events in E and returns the required time Δt.

3.1.1 Parameters

One of the goals during the design stage was to reduce the number of parameters as much as possible. Table 3.1 shows the parameters to be adjusted, which will depend on the computational requirements of the system on which the event-based algorithm is running. It should be noted that parameters are presented here just for reference, since they will be extensively explained in later chapters.

The parameter setting depends on the desired ASAP behaviour, always assuming a compromise between latency and robustness. Algorithms with similar complexity will require similar parameter values, so an ASAP tuning is generally valid for several event-based algorithms.

3.1.2 Outer Loop: Event Filtering

Event filtering is a preprocessing stage in which a percentage of the generated events will be randomly removed. To do this, a random value is generated from a

Table 3.1 ASAP parameters.

	Description	**When increases**	**When decreases**	**Typical value**	**More details in**
N_R	Size of the window used to compute event rate	Better estimation, slower updating	Worse estimation, faster updating	$\sim$1000 events	Chapter 4
γ_{min}	Minimum value for event filtering	Fewer removed events	More removed events	$\sim$ [0.1, 0.4]	
γ_{max}	Maximum value for event filtering	More removed events	Fewer removed events	$\sim$ [0.7, 1]	
K_p	Proportional gain to adjust event packing	Higher response velocity, more risk of instability	Lower response velocity, less risk of instability	-	Chapter 5
$\Delta \bar{t}_{ref}$	Desired responsiveness level	Less responsiveness	More responsiveness	$\sim$ [0.1, 0.9]	
S_{min}	Minimum packet size before sending	Less responsiveness	More responsiveness	$\sim$ [1, 100] μs	
S_{max}	Maximum packet size before sending	Less responsiveness	More responsiveness	$\sim$ [10^3, 10^5] μs	

normal distribution that is compared to a threshold.

The filter is adjusted through the γ parameter. The lower the γ, the higher the deleted events percentage. A value of $\gamma = 1$ implies that no event is removed, while with $\gamma = 0$ all events would be discarded. Rodríguez-Gómez et al. demonstrated in [63] –where they used a constant γ value– that a high percentage of events could be filtered out without losing relevant information.

The γ value is automatically adapted according to the input event rate –information from scenario– and the processing requirements –information fed back from the algorithm–. γ will range from a maximum γ_{max} to a minimum γ_{min} value that can be adjusted.

This preprocessing filter will be extensively detailed in Chapter 4.

3.1.3 Inner Loop: Event Packing

The main module of ASAP is the event packer. The generated events are stored in a packet and sent to the algorithm when a certain time condition is satisfied. When a packet is sent, a new packet will start to be filled in.

The time condition to be verified for sending a packet E_k varies adaptively according to the computational time the algorithm needed to compile the previous packet E_{k-1} –i.e., Δt, which is fed back from the algorithm–.

This module will be further detailed in Chapter 5.

3.1.4 ASAP Communication Structure

ASAP runs as a ROS node and communicates with the event-based algorithm through messages (Figure 3.2) published in a topic to which the algorithm must subscribe.

Event packet messages consist of a timestamp –by convention, packets are tagged with the last-added event timestamp–, the sensor dimensions –width and height– and an event array –i.e., the packet–.

Each event within a packet is a message composed of a timestamp, the position of the event in image plane coordinates –x and y, being (0,0) the upper left corner of the sensor– and the event polarity —positive when the lighting change is an increase and negative when a decrease–.

Time feedback is also communicated using a ROS message. This value is sent by the algorithm and received by ASAP.

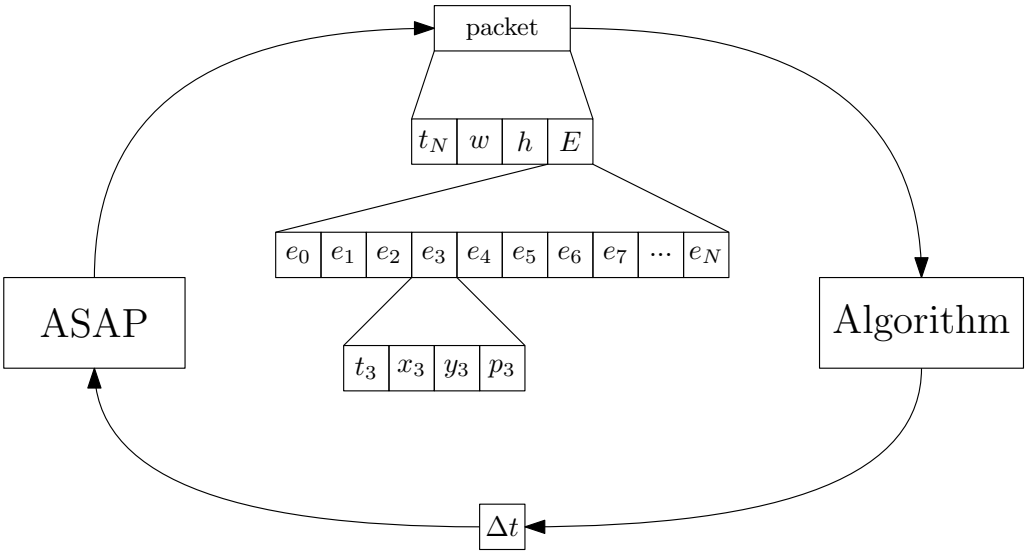

Figure 3.2 Communication through messages in ASAP. See Table 3.2 for notation.

Table 3.2 Figure 3.2 notation.

t_k	Time instant in which event e_k was generated
x_k, y_k	Position in which event e_k was generated
e_k	k-th event in E
E	Event packet
h, w	Sensor height and width
Δt	Time required to process E

3.2 Event-Based Algorithm Connection: BRIDGE

One of the initial problems with ASAP was the need to modify the algorithm to return the temporal cost of processing each received packet. Although this is not a very complex modification, it reduces the ASAP modularity. BRIDGE was designed to solve this problem.

BRIDGE is an interface that modifies the event handler –the callback function that is executed every time a packet is received– to calculate the time that the algorithm has taken to process each event in the packet (see Figure 3.3). So, actually, ASAP sends event packets to BRIDGE and the algorithm only has to subscribe to it for acquiring the events.

The time outputted by BRIDGE is normalized in order to return a value between

0 and 1 (where 0 means that the temporary cost is the lowest possible and 1 the highest). To normalize Δt, expression in Equation 3.1 is used, where Δt_{min} and Δt_{max} are respectively the lowest and highest Δt values ever obtained and $\bar{\Delta t} \in [0,1]$ is the normalized value.

$$\bar{\Delta t} = \frac{\Delta t - \Delta t_{min}}{\Delta t_{max} - \Delta t_{min}} \tag{3.1}$$

To prevent high values of Δt from leading to poor normalization conditioning, a forgetting factor α can be used in order to decrease the value of Δt_{max} over time, as shown in Equation 3.2. This is more necessary the higher the variance of the algorithm temporal cost.

$$\Delta t_{max} := (1-\alpha) \cdot \Delta t_{max} \tag{3.2}$$

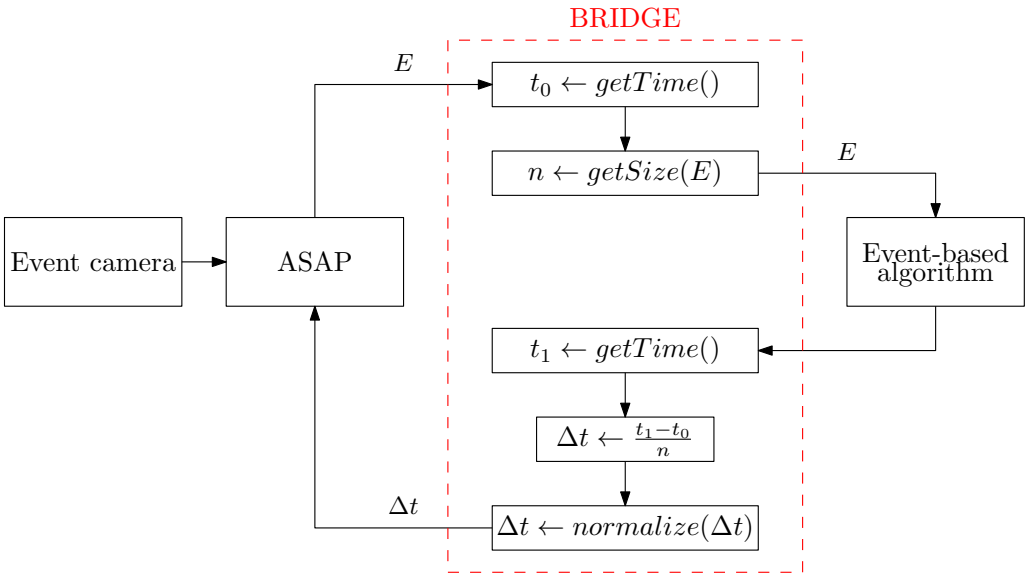

Figure 3.3 BRIDGE simplified scheme.

Algorithm 1 shows this interface operation, where E is an event packet and Δt is the average temporal cost per event in E.

Algorithm 1: BRIDGE Interface

Input : E
Output : Δt
Parameters : α ▷ Optional
Initialization : $\Delta t_{max} \leftarrow 0$, $\Delta t_{min} \leftarrow \infty$ ▷ Execute only the first time

```
n ← getSize(E)
t_0 ← getCurrentTime()
callAlgorithm(E)
t_1 ← getCurrentTime()
Δt ← (t_1 - t_0) / n
if Δt > Δt_max then
    Δt_max ← Δt
else if Δt < Δt_min then
    Δt_min ← Δt
end
if Δt_min = Δt_max then
    Δt ← 0
else
    Δt ← (Δt - Δt_min) / (Δt_max - Δt_min)
end
Δt_max ← (1-α) · Δt_max                    ▷ Optional
```

3.3 Conclusions

This chapter has intended to provide a general overview of how ASAP works. It has been used as an introduction to the next two chapters, where the concepts summarized here will be explained in further detail.

In addition, thanks to BRIDGE, ASAP can be presented as a modular method easy to couple to different algorithms. The aim of this connection is to make ASAP an easy-to-use alternative for any user in the event perception research community.

4 Event Filtering Module

This chapter presents the event preprocessing. This first stage (Figure 4.1) reduces the workload of the system on which ASAP is executed, so it is an indispensable stage for systems with low computing capacities.

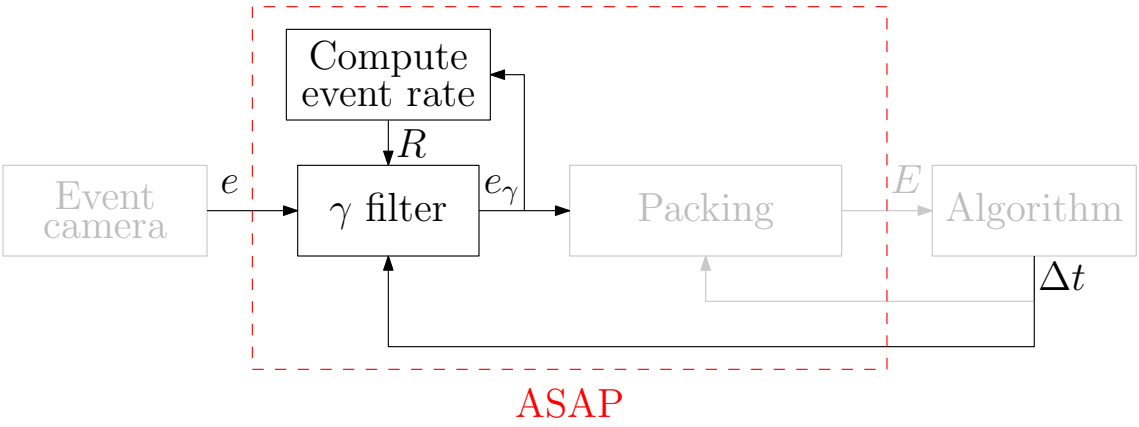

Figure 4.1 ASAP event filtering module.

The events generated by the silicon retina are acquired through the AER protocol by ASAP. To decrease the computational cost, not all events will be processed. For an event to be processed it must pass the γ filter, which randomly discards them.

It is very important to avoid excessive filtering –i.e., removing too much relevant information–. Ideally, all deleted events should be redundant information from the scene.

4.1 Effect of Event Filtering

The implemented filter is based on the procedure proposed by Rodrígez-Gómez et al. in [63]. The method they presented uses a randomly generated value according to a normal distribution that is compared with a constant $\gamma \in [0, 1]$ threshold (see Algorithm 2, where e is the event to be evaluated).

Figure 4.2 shows the results obtained by applying different values of γ. It is possible to check the lower γ the lower the number of events in the image –i.e., the lower the information from scene–.

Algorithm 2: Constant-valued γ filtering

Input : e
Output : e or no output
Parameters : γ

```
r ← generateRandomValue(0,1)
if r < γ then
    pass(e)
else
    remove(e)
end
```

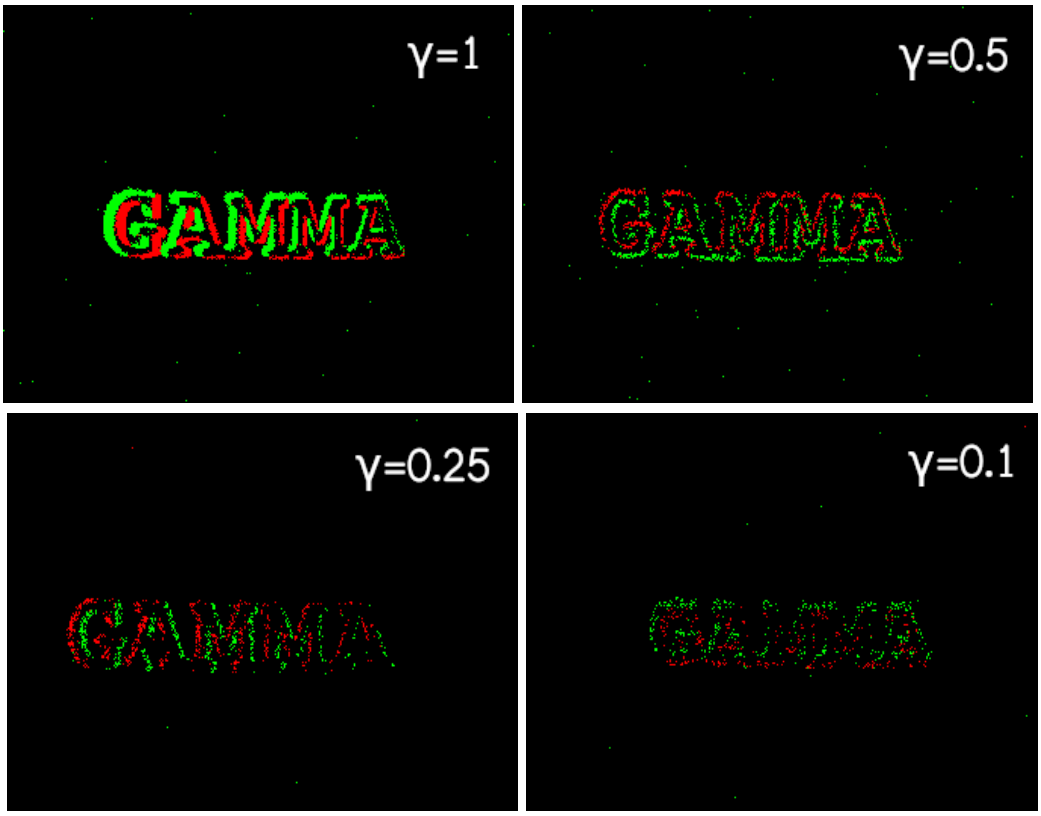

Figure 4.2 Event images for different γ values.

4.2 Event Filter Operation

The advantage of the designed filter over the method proposed in the previous section is that it is capable of varying γ according to the computational needs of the event-based algorithm.

The γ value will vary between γ_{min} and γ_{max} values depending on the event rate R –i.e., events generated per second– as expressed in Equation 4.1.

$$\gamma = \gamma_{max} - \bar{R} \cdot (\gamma_{max} - \gamma_{min}) \tag{4.1}$$

The $\bar{R}$ value in Equation 4.1 denotes the normalized event rate (see Equation 4.2). To make the normalization more robust, a forgetting factor α that decreases R_{max} over time can be established.

$$\bar{R} = \frac{R - R_{min}}{R_{max} - R_{min}} = \frac{R}{R_{max}} \tag{4.2}$$

The γ_{min} and γ_{max} values are defined by the user to control the maximum and minimum amount of events to be deleted. To take into account the computational cost Δt of the vision algorithm, the γ_{max} value can be modified by ASAP. This way, when the algorithm is overloaded, γ_{max} will decrease to reduce the event rate and vice versa. However, the range of γ_{max} values can still be controlled by the user through γ_{max}^{BOT} and γ_{max}^{TOP} (so that $\gamma_{max} \in [\gamma_{max}^{BOT}, \gamma_{max}^{TOP}]$ as shown in Equation 4.3).

$$\gamma_{max} = \gamma_{max}^{TOP} - \Delta\bar{t} \cdot (\gamma_{max}^{TOP} - \gamma_{max}^{BOT}) \tag{4.3}$$

Figure 4.3 presents the behaviour of γ as a function of the normalized event rate $\bar{R}$ and the normalized temporal cost of the algorithm Δt.

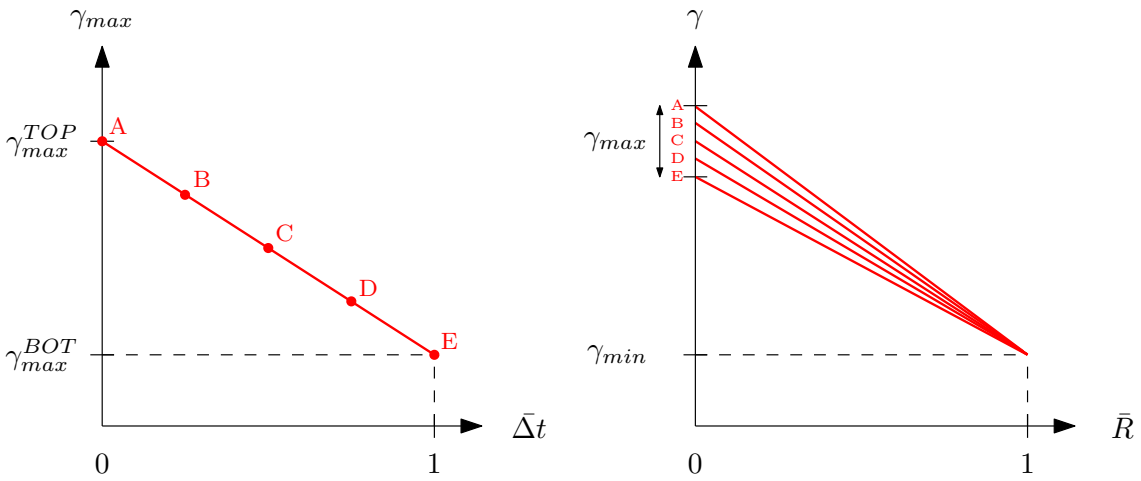

Figure 4.3 Functions $\Delta\bar{t}$ vs. γ_{max} (left) and $\bar{R}$ vs. γ (right).

4.2.1 Event Rate Computation

The event rate indicates how the number of events varies over time (see Equation 4.4, where n is the number of events received). In order to implement the event rate computation (see Algorithm 3), a window of fixed size N_R (Figure 4.4) will be used, thus the rate can be calculated using the expression in Equation 4.5.

$$R(t) = \frac{dn}{dt} \tag{4.4}$$

$$R_k = \frac{\Delta n}{\Delta t} = \frac{N_R}{t_k - t_{k-N_R+1}} \tag{4.5}$$

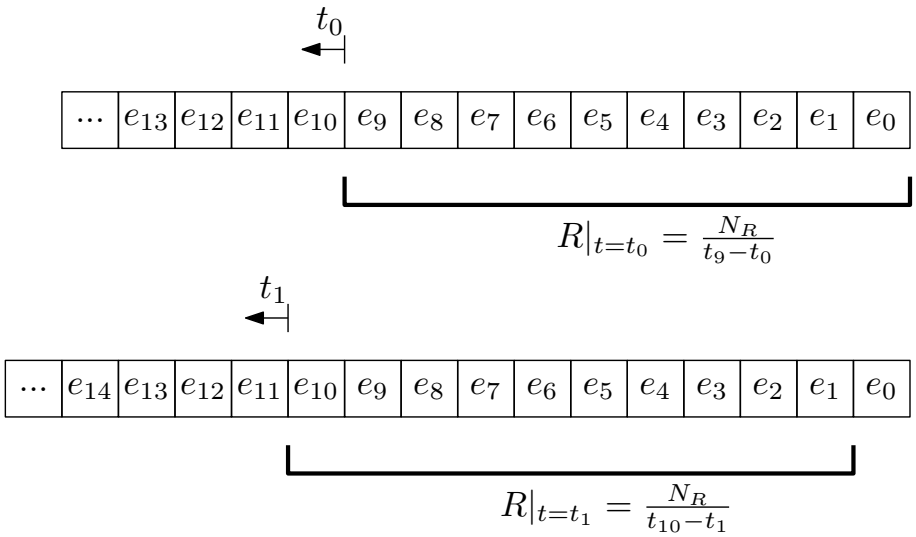

Figure 4.4 Example of event rate computation for $N_R = 10$.

4.2.2 Parameters

Table 4.1 shows the parameters used by the filtering method. These are parameters whose value must depend on the computing capacity of the system on which ASAP is running.

Table 4.1 Filtering module parameters.

γ_{min}	Minimum γ value that can be achieved
$\gamma_{max} = [\gamma_{max}^{BOT}, \gamma_{max}^{TOP}]^\top$	Maximum γ value that can be achieved
N_R	Window size used to compute event rate
α (optional)	Forgetting factor for R_{max}

Algorithm 3: Event rate computation

Input : e
Output : R
Parameters : N_R
Initialization : $R \leftarrow 0$, $R_{list} \leftarrow$ emptyList()

$R_{list} \leftarrow$ insert(R_{list}, e)
if length(R_{list}) > N_R **then**
  $R_{list} \leftarrow$ removeOldestItem(R_{list})
  $r_n \leftarrow$ getNewestItem(R_{list})
  $r_o \leftarrow$ getOldestItem(R_{list})
  $t_r \leftarrow$ getTime(r_n) - getTime(r_o)
  $R \leftarrow N_R$ / t_r
end

4.2.3 Implementation

Algorithm 4 presents the γ calculation. This allows the constant-valued filter introduced above to be modified to obtain an adaptive filtering (see Algorithm 5).

Algorithm 4: γ computation for event filtering

Input : R, $\Delta\bar{t}$
Output : γ
Parameters : γ_{min}, γ_{max}^{BOT}, γ_{max}^{TOP}, α
Initialization : $R_{max} \leftarrow 0$ ▷ Execute only the first time

$\gamma_{max} \leftarrow \gamma_{max}^{TOP} - (\gamma_{max}^{TOP} - \gamma_{max}^{BOT}) \cdot \Delta\bar{t}$
if R > R_{max} **then**
  $R_{max} \leftarrow R$
end
$\bar{R} \leftarrow R / R_{max}$
$\gamma = \gamma_{max} - (\gamma_{max} - \gamma_{min}) \cdot \bar{R}$
$R_{max} \leftarrow (1\text{-}\alpha) \cdot R_{max}$ ▷ Optional

4.3 Computational Capacity vs. Filtering

It should be noted that this processing stage is designed specifically for systems with limited computing constraints (e.g., on-board computers). However, by adjusting the parameters, it is possible to reduce or even eliminate the filtering of acquired events. This implies no loss of information.

Algorithm 5: Variable-valued γ filtering

Input : e, γ
Output : e or no output

```
r ← generateRandomValue(0,1)
if r < γ then
    pass(e)
else
    remove(e)
end
```

It is up to the user to decide how much information to remove for the sake of a faster processing. For example, on desktop computers it will usually not be necessary any filtering.

4.4 Preliminary Results

This section presents some results obtained when testing the event filtering. The value of R and γ for a low event rate scenario (Figure 4.5) and a high event rate scenario (Figure 4.6) are shown. Values $\gamma_{min} = 0.3$ and $\gamma_{max} = 0.9$ were used. Figure 4.7 shows the result obtained by allowing ASAP to vary γ_{max} between $\gamma_{max}^{BOT} = 0.7$ and $\gamma_{max}^{TOP} = 0.9$.

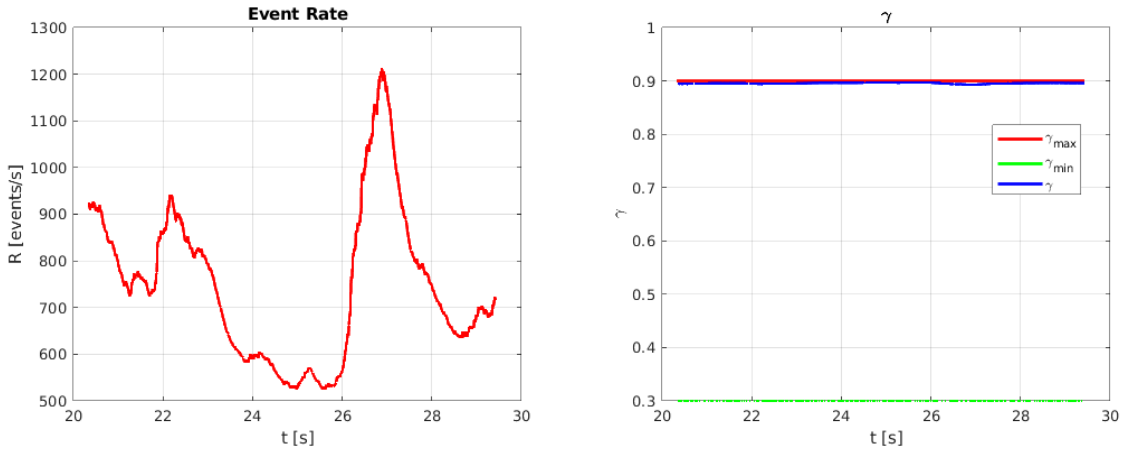

Figure 4.5 γ filter preliminary results in low event rate scenario. Constant γ_{max}.

4.5 Conclusions

Although information loss is generally never desirable in a perception system, removing redundant information is a perfect way to dedicate all processing power to the most useful data.

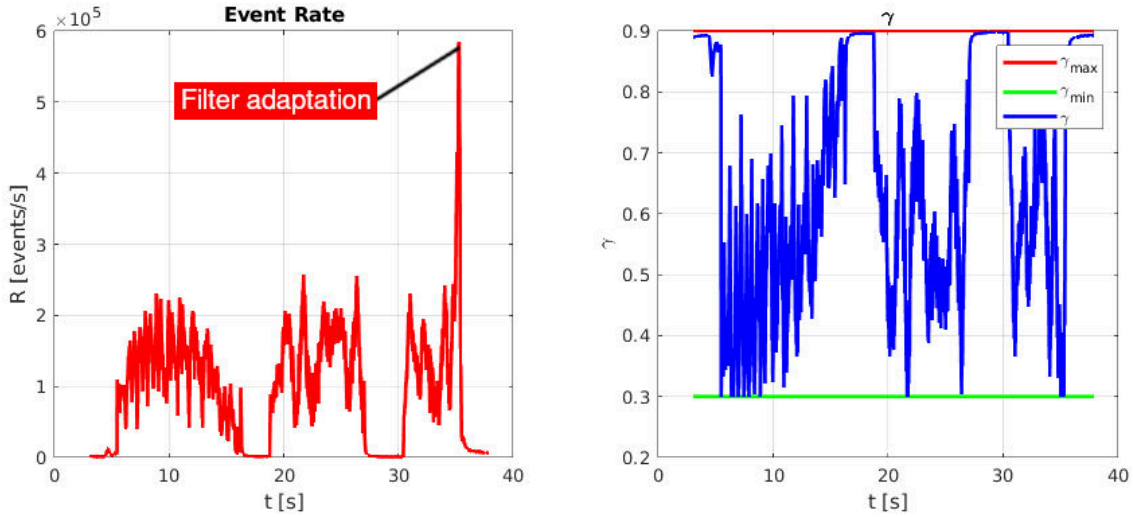

Figure 4.6 γ filter preliminary results in high event rate scenario. Constant γ_{max}.

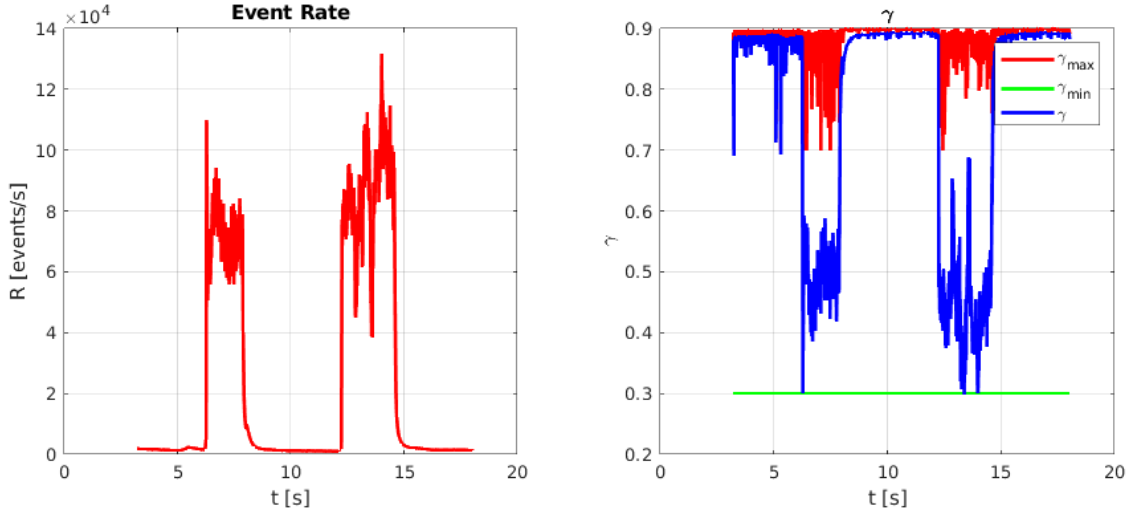

Figure 4.7 γ filter preliminary results in high event rate scenario. Variable γ_{max}.

The main conclusion of this chapter is that it is possible to automatically adapt the filtering by only using information from the scene.

Events not removed by the filter will be sent to the packing module, which will be described in the next chapter.

5 Event Packing Module

This chapter will detail the operation of the main ASAP module: the event packer (Figure 5.1). This stage allows to adjust the events transmission to achieve an efficient and highly responsive operation of the event-based vision algorithm.

Furthermore, the parameters of this adaptive scheme and its effect on processing will be presented. Some notions on how to adjust these parameters to achieve different processing behaviours will also be given.

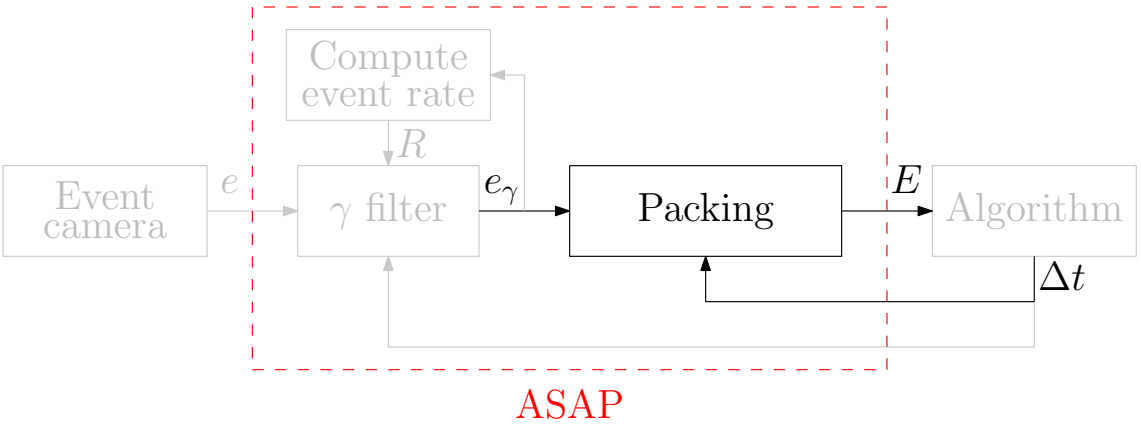

Figure 5.1 ASAP event packing module.

5.1 Effect of Event Packing on Latency

Event-by-event processing allows very low latency – i.e., real-time operation –. However, high-speed movements will cause millions of events per second, so being able to process all of them will require a very high computational capacity.

On the other hand, event-packet processing allows near real-time performance but without taking full advantage of the event camera's time resolution.

Comparing both alternatives, it can be concluded that the higher the event packet,

the higher the latency between event generation and processing but the lower the computational requirement. See Figure 5.2, corresponding to experiments in which packets of 100 events generated a latency around 0.2 - 0.3 milliseconds, packets of 1000 events generated a latency around 2 - 3 milliseconds, and packets of 10000 events generated a latency around 30 milliseconds

Therefore, it will be necessary to use small packets when not generating many events per second –to achieve real time processing– and large packets when generating many events –to avoid overflowing–.

5.2 Event Packer Operation

In order to set the packet size S, ASAP employs the computational cost Δt fed back from the event-based algorithm using the expression from Equation 5.1. Although the expression is based on that of a proportional incremental control with error $\Delta\bar{t} - \Delta\bar{t}_{ref}$, it should be clear that the aim is not to stabilize the feedback signal according to a reference, since it is assumed that the scenario will be in a continuous change and a constant $\Delta\bar{t}_{ref}$ value cannot be achieved.

$$S := S + K_p \cdot (\Delta\bar{t} - \Delta\bar{t}_{ref}) \tag{5.1}$$

The computed size can be used as a threshold of number of events per packet or as a threshold of time in the packet –i.e., time of the most recent event minus time of the oldest event)–. The settings of K_p and $\Delta\bar{t}_{ref}$ will vary depending on the type of threshold used. From now on, a temporary threshold will be used, since it has been proven to give better results (see Chapter 7).

The threshold will move between the values S_{min} and S_{max}, used to set the desired minimum and maximum processing responsiveness.

5.2.1 Parameters

Table 5.1 shows the parameters used by the packing method. These are parameters whose value must depend on the event-based algorithm complexity and the responsiveness to be achieved.

Table 5.1 Packing module parameters.

K_p	Proportional gain to adjust event packing
$\Delta\bar{t}_{ref}$	Desired responsiveness level
S_{min}	Minimum allowed size
S_{max}	Maximum allowed size

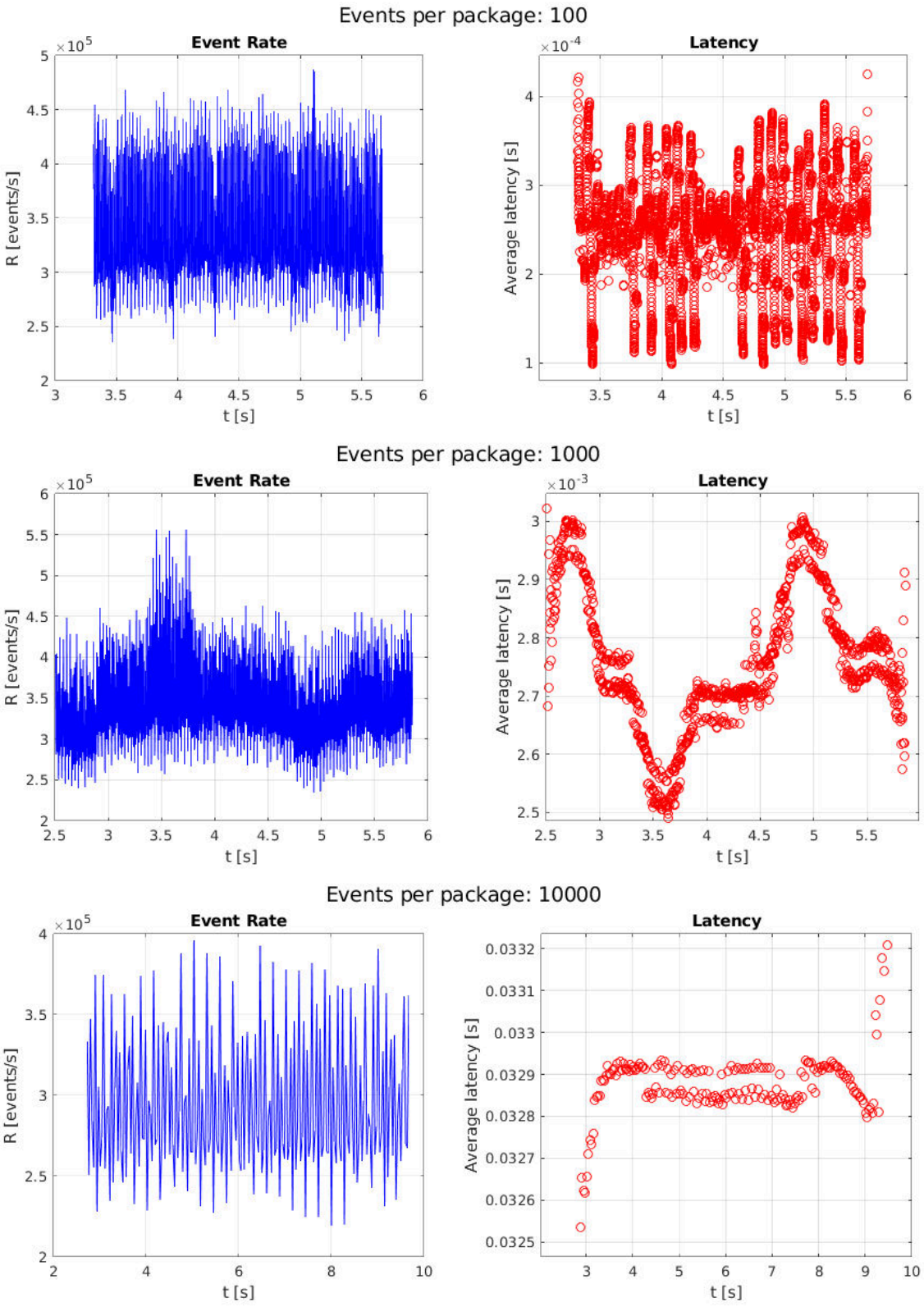

Figure 5.2 Time difference between event generation and processing for different packet sizes.

5.2.2 Proportional Gain Adjustment

The proportional gain defines how quickly the threshold is updated. It is difficult to define a typical value for this variable, since it will depend on the nature of the event-based algorithm. Even so, similar complexity algorithms may use the same K_p value.

For relatively low values, the threshold will not be updated fast enough to fit the real-time environment. Otherwise, relatively high values will cause a rapid saturation of the threshold (see Figure 5.3).

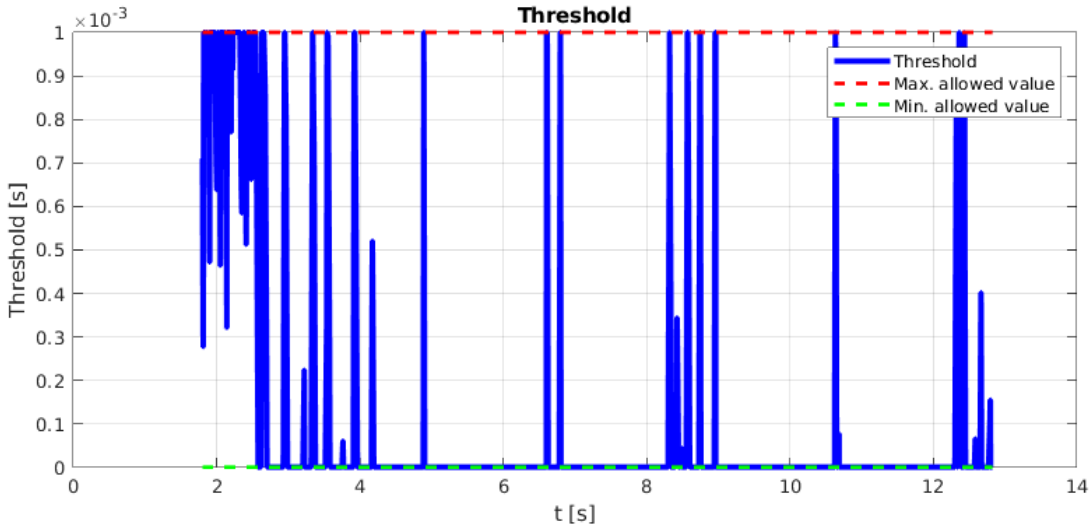

Figure 5.3 Saturation caused by a very high K_p value.

Generally speaking, increasing K_p will lead to a faster update of the threshold and therefore a higher adaptive capacity and decreasing K_p will lead to less risk of adjustment instability.

5.2.3 Responsiveness Level Adjustment

The value $\Delta\bar{t}_{ref}$ allows to determine the average value of the desired algorithm temporal cost, where $\Delta\bar{t}_{ref} = 1$ is the longest time taken for processing and $\Delta\bar{t}_{ref} = 0$ is the shortest.

As mentioned above, this value should not be understood as a reference to be reached, but as a parameter that defines the algorithm responsiveness to be achieved. This parameter makes it possible to adjust the compromise between processing speed and robustness. As can be proven in Equation 5.1, small values of $\Delta\bar{t}_{ref}$ imply higher tendency to increase the threshold, and therefore, large packet generation –i.e., low latency but high robustness–. In contrast, large values imply higher tendency to decrease the threshold, and consequently, smaller packets or even event-by-event processing –i.e., high latency but low robustness–.

5.2.4 Implementation

Algorithm 4 –where e is the input event from γ-filter, E is the current event packet, and Δt is the temporal cost per event of the previous packet– corresponds to the ASAP packing module implementation.

Algorithm 6: Event packing

Input : e, $\Delta\bar{t}$
Output : E or no output
Parameters : S_{min}, S_{max}, K_p, $\Delta\bar{t}_{ref}$
Initialization : E ← emptyList() ▷ Execute only the first time

```
1  E ← insert(E, e)
2  S ← S + K_p · (Δt̄ - Δt̄_ref)
3  if S > S_max then
4  |  S ← S_max
5  else if S < S_min then
6  |  S ← S_min
7  end
8  e_n ← getNewestItem(E)
9  e_o ← getOldestItem(E)
10 s ← getTime(e_n) - getTime(e_o)
11 if s > S then
12 |  sendToAlgorithm(E)
13 |  E ← emptyList()
14 end
```

5.3 Preliminary Results

This section presents some results obtained when testing the event packing. Values $K_p = 10$, $\Delta\bar{t}_{ref} = 0.1$, $S_{min} = 1\mu s$ and $S_{max} = 1ms$ have been used. Figure 5.4 shows the results for a low event rate scenario, while Figure 5.5 corresponds to a high event rate scenario.

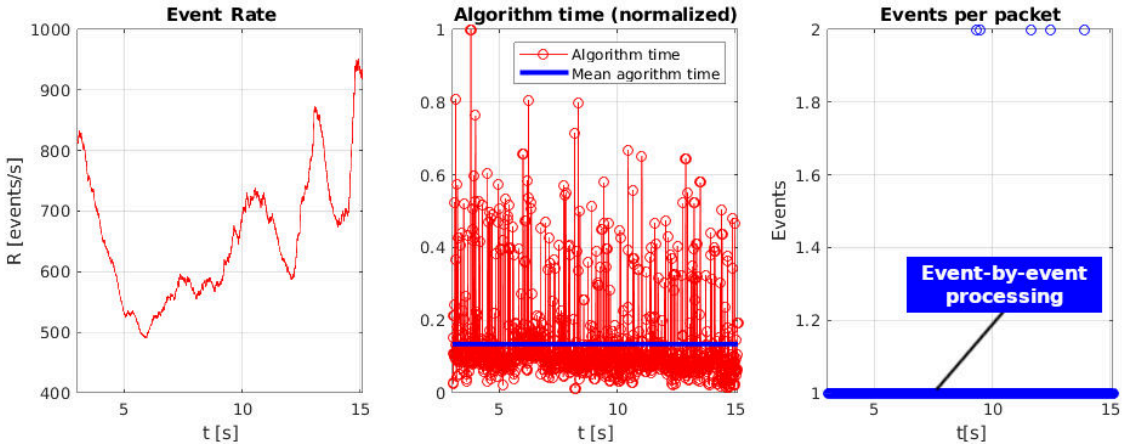

Figure 5.4 Event packing preliminary results in low event rate scenario.

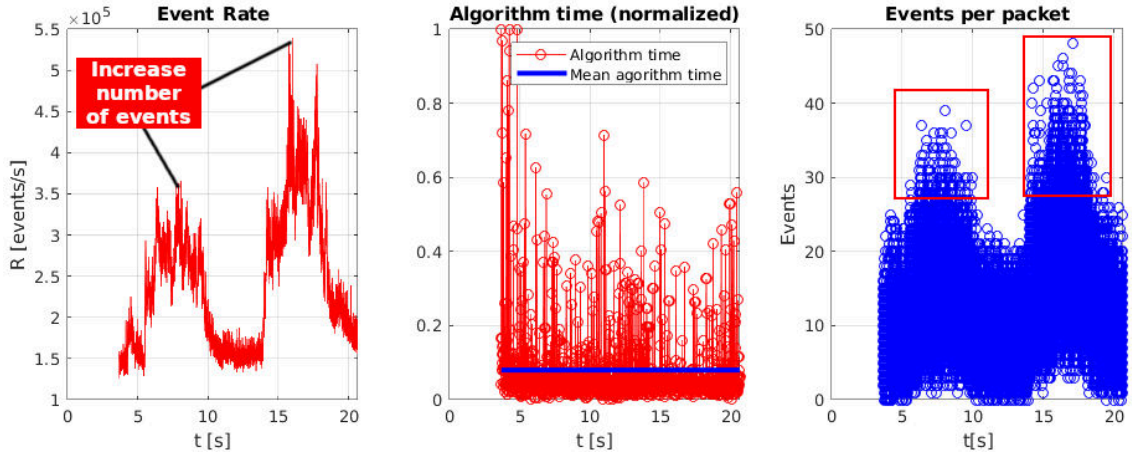

Figure 5.5 Event packing preliminary results in high event rate scenario.

5.4 Event-by-Event Processing using ASAP

It is worth noting that through an appropriate parameter setting, it is possible to get ASAP to send events to the algorithm one by one. It is also possible to adjust a general event-by-event processing that switches to packet-based processing only at certain times, when there is a large generation of events per second.

This feature makes ASAP capable of adapting to various applications and devices with different computational capabilities.

5.5 Conclusions

This chapter has tried to show the relevance of event packing on the event-based vision algorithm connected to ASAP. Through the parameter adjustment, it is possible to achieve different behaviours that emphasize different features (e.g., computational efficiency, robustness against overflowing, low latency...), as well as operations that combine some of these features.

This chapter, in conjunction with the two previous ones, constitutes the description of the designed adaptive scheme for asynchronous event processing. Appendix 1 contains the complete implementation of the described method.

6 Experimental Validation

This chapter presents the experimental evaluation performed to validate the method proposed in this document. Several tests were carried out, both off-board –for parameters setting– and on-board a multirotor. The three most significant experiments –one off-board and two on-board– will be shown here.

First, an off-board test is presented. A *dummy* algorithm was used to close the feedback loops and emulate a vision algorithm whose complexity increases over time –for each event there is an increasing delay–. This experiment was employed to verify if ASAP is able to work in a wide range of temporal cost.

After that, two experiments performed to test ASAP on-board an aerial robot are presented. An event-based clustering algorithm [63] and an event-based FAST corner detector [44] were used.

6.1 Off-Board Experiments

Different tests were performed to adjust the ASAP parameters. The selected values are shown in Table 6.1 –no forgetting factors were established, since the experiments will have a not very long duration–. The same values will be used for the different on-board tests to demonstrate that a same setting can work for algorithms with similar complexity.

Table 6.1 Parameters for experimental validation.

γ Filtering				Event Packing			
γ_{min}	γ_{max}^{BOT}	γ_{max}^{TOP}	N_R	K_p	$\Delta\bar{t}_{ref}$	S_{min}	S_{max}
0.3	0.7	0.9	1000	5	0.15	1 μs	1 ms

In order to validate the correct operation of the method, it was implemented an algorithm whose computational cost increased each time it received a packet (see Algorithm 7). The goal of this test was to verify the adaptive behaviour of ASAP. The obtained results will be analysed in the following chapter.

Although it is not a realistic case –no algorithm temporal cost experiences a constant variation over time–, this evaluation will allow to verify if ASAP can be adapted according to a variable temporal cost.

Algorithm 7: *Dummy* algorithm for testing

Input : E
Initialization : $t \leftarrow 0$ ▷ Execute only the first time
1 $t \leftarrow t + 10^{-6}$ ▷ Increase 1 microsecond
2 waitSeconds(t)

6.2 On-Board Experiments

The experimental setup consists of a DAVIS346 event camera on-board a DJI Flamewheel F450 with a PixRacer autopilot (Figure 6.1). ASAP was running on a low-cost Khadas VIM3 single-board computer on top of the UAL [58] using ROS Kinetic Kame and the PX4 low-level controller. Multirotor flights were conducted inside an OptiTrack testbed.

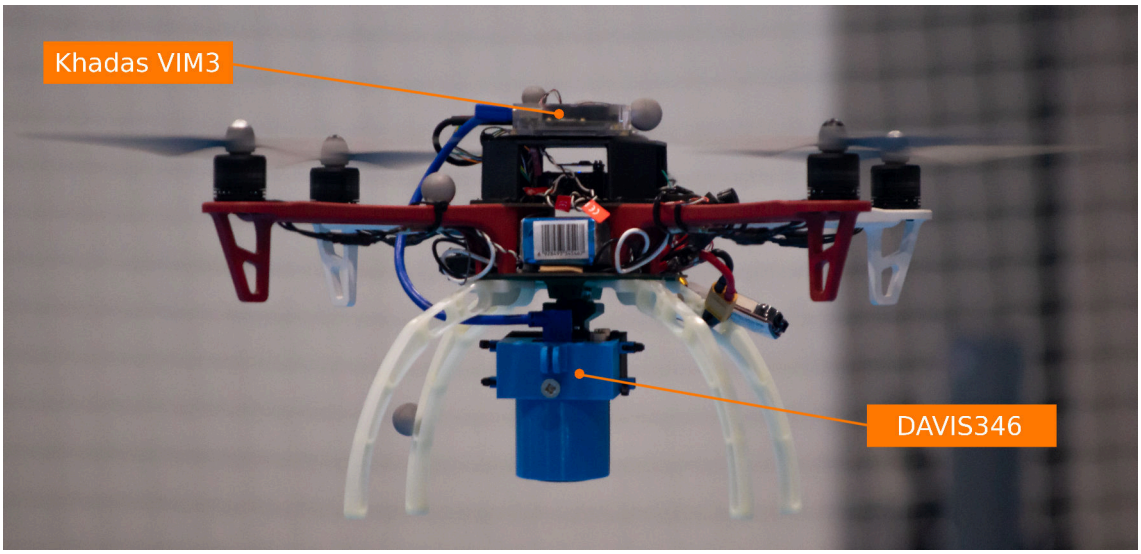

Figure 6.1 Aerial robot based on DJI Flamewheel F450 equipped with a DAVIS346 event camera and a Khadas VIM3 single-board computer.

6.2.1 Clustering

The first tested algorithm was the event-based clustering proposed by Rodríguez-Gómez et al. in [63]. The experimental scenario consisted of a set of pipes placed vertically over which the multirotor flew (see Figure 6.2). From the events generated by the multirotor movement, the algorithm was able to group the pipes into clusters (Figure 6.3).

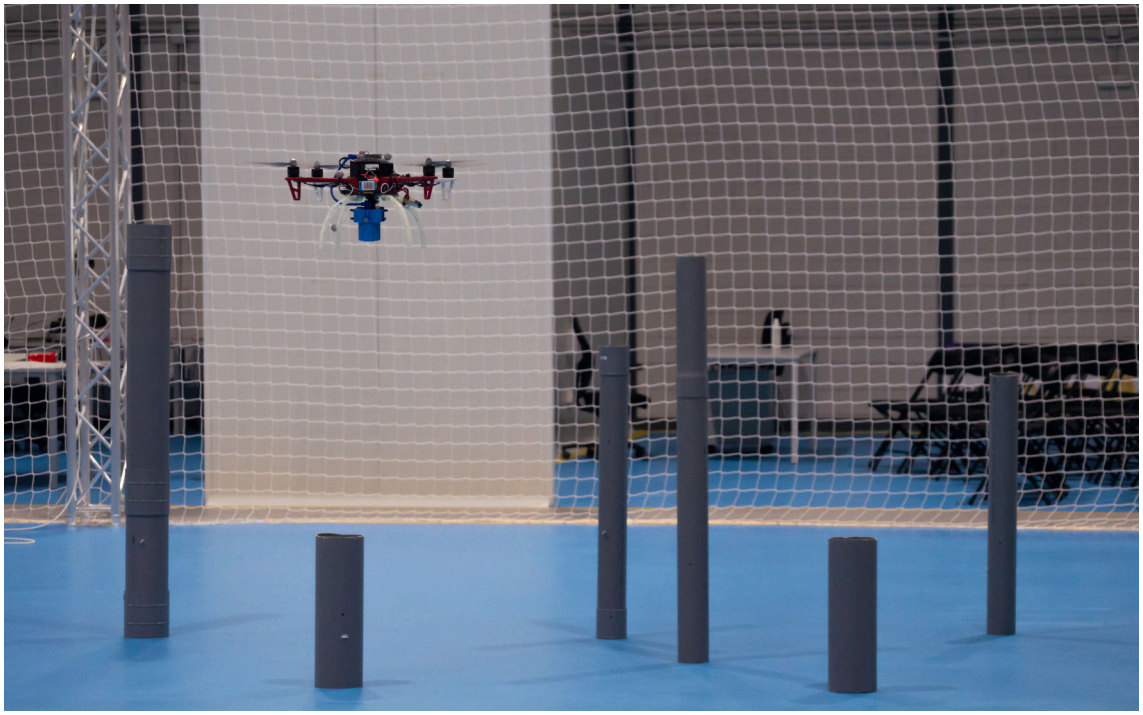

Figure 6.2 Experimental setup for event-based clustering test.

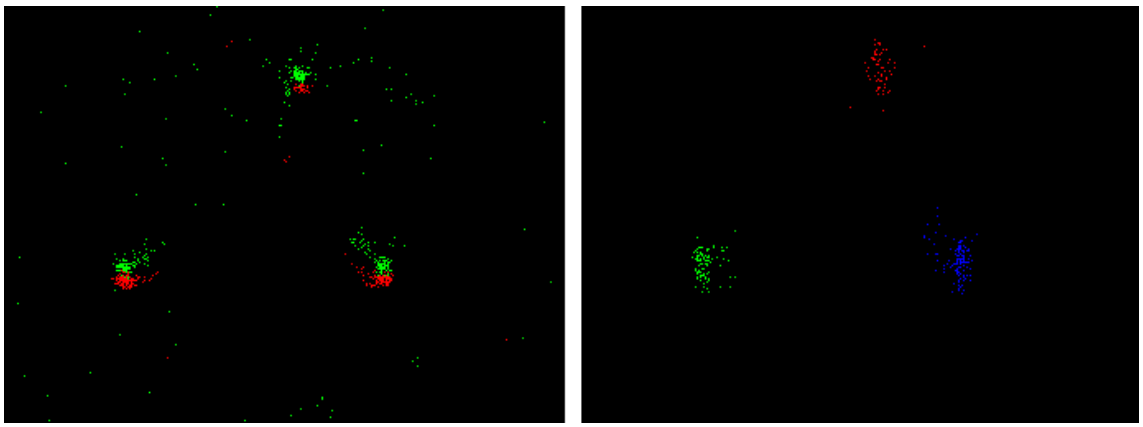

Figure 6.3 Event-based clustering result. Generated events (left) and corresponding clusters (right).

6.2.2 Corner Detector

The second algorithm to be tested was the event-based FAST corner detector presented by Mueggler et al. in [44]. The scenario of the experiment consisted of a grid formed by three horizontal and three vertical lines (see Figure 6.4). Figure 6.5 shows some results obtained by the corner detector.

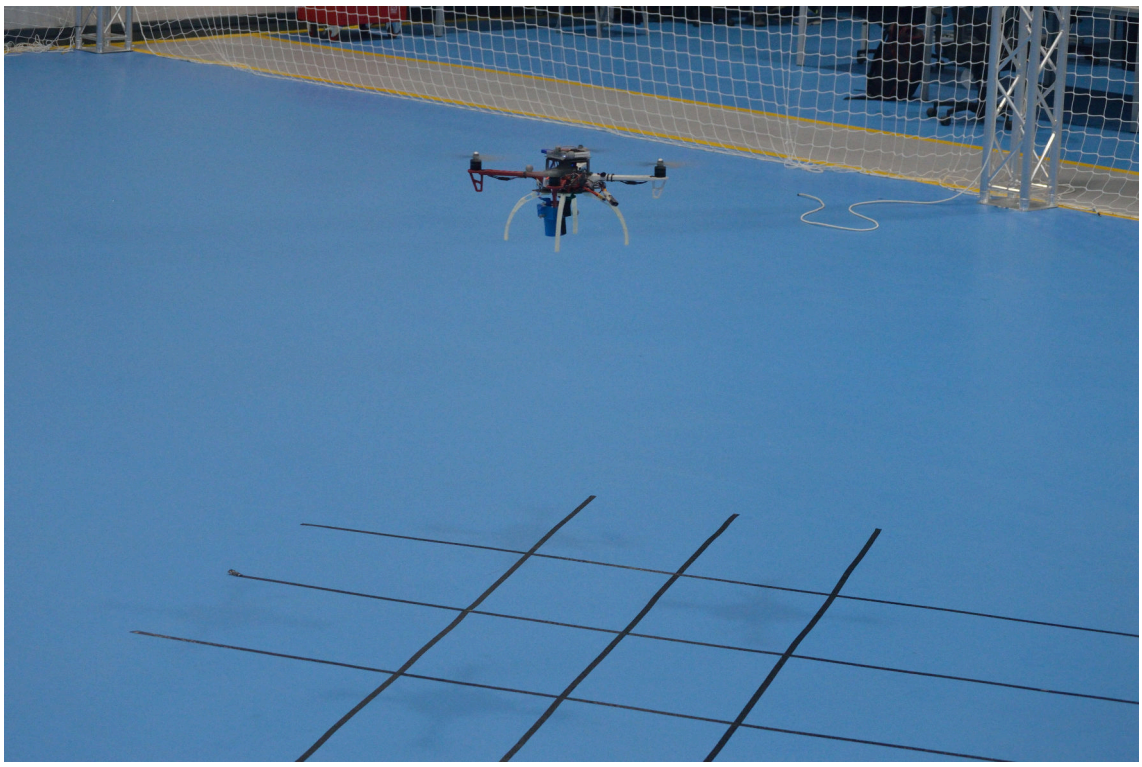

Figure 6.4 Experimental setup for event-based FAST corner detector test.

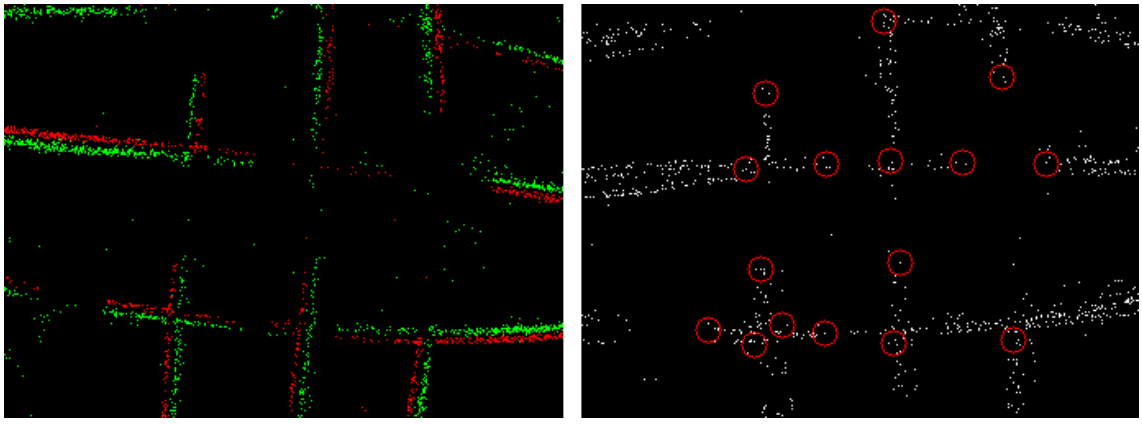

Figure 6.5 Event-based FAST corner detector result. Generated events (left) and detected corners (right).

6.3 Conclusions

This chapter has sought to describe the experimental validation performed to verify and evaluate ASAP. The analysis of the logged results will be detailed in Chapter 7. Two algorithms with quite different processing performance have been used to check the adaptive nature of ASAP.

It is worth noting that the results of the event-based vision algorithms (Figures 6.3 and 6.5) are presented here by way of illustration, since neither the accuracy nor the efficiency of the algorithm is an objective for this validation.

7 Result Analysis

The results obtained after the experiments described in the previous chapter will be analysed here. From this analysis it will be possible to validate the efficient and robust operation of ASAP.

Three analyses are presented, corresponding to the three experiments performed: an off-board test, the on-board event-based clustering and the on-board event-based FAST corner detector.

7.1 Off-Board Experiment Results

Figure 7.1 shows the results obtained by placing the event camera in front of a a low-contrast and low-dynamic scene. An algorithm whose computational cost increases over time was used (see Chapter 6).

Note that as the processing cost increases, the length of the packets –i.e., the number of events per packet– increases. For this reason, the temporal cost per event (see top right graphic in Figure 3) remains within a constant value range.

The γ value is adapted according to the event rate R. Due to the nature of the scenario, γ does not reach very low values –although the lower limit γ_{min} was 0.3, the minimum reached is approximately 0.8–.

These results validate the adaptive nature of ASAP, both to the environment and to the processing of the algorithm. This behaviour is desired for asynchronous event processing, so it can be stated that one of the main objectives of this project has been fulfilled.

7.2 Clustering Experiment Results

Graphics in Figure 7.3 correspond to the event-based clustering test (Figure 7.2). The peaks that appear in the event rate –top left graphic– correspond to moments

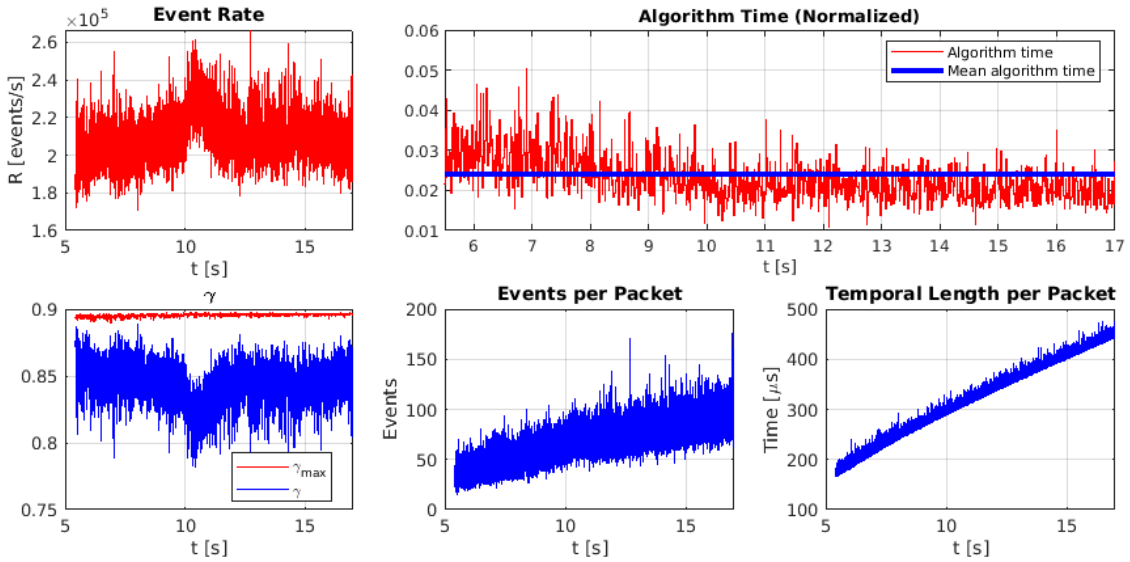

Figure 7.1 Off-board experiment results. From top-left: event rate, normalized algorithm temporal cost (from 0 to 1), γ value, events per packet and time difference between the newest and the oldest event in each packet.

in which a large number of events are generated due to an abrupt multirotor movement or to the existence of many pipes in the field of view of the camera.

Each of these peaks corresponds to a γ filter peak –see bottom left graphic–. The filtering is activated in case of an excess of events per second to avoid the saturation of the algorithm.

In terms of processing temporal cost, it can be seen that the peaks also appear –top right graphic–, but quite attenuated. ASAP has managed event stream to homogenize the temporal cost in values generally below 0.4. Remember that the temporal cost is represented normalized –i.e., 0 is the shortest time required by the clustering to process an event and 1 the longest–.

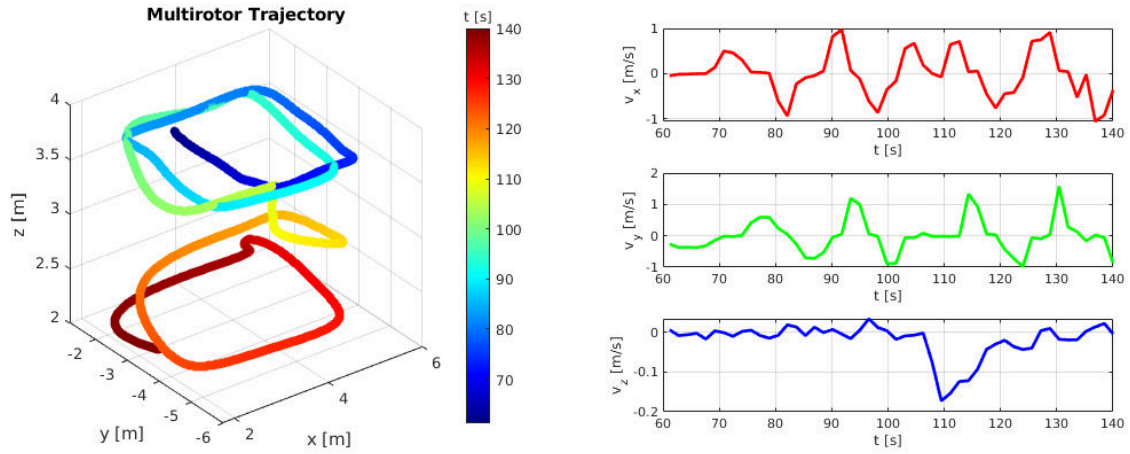

Figure 7.2 Multirotor trajectory during experiment. Position (left) and velocity (right) over time.

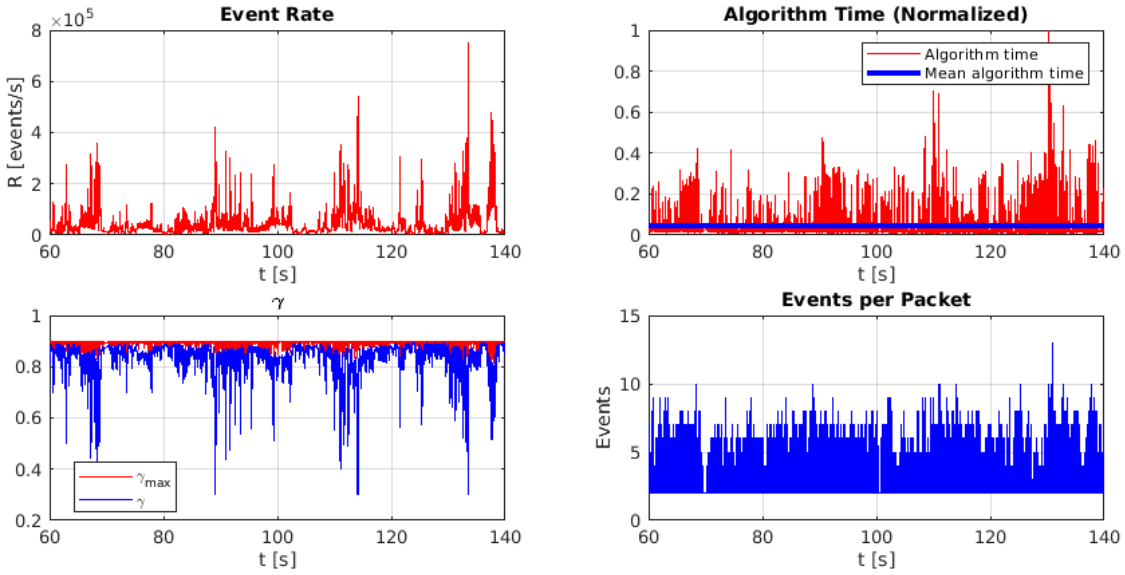

Figure 7.3 Event-based clustering experiment results. From top-left: event rate, normalized algorithm temporal cost (from 0 to 1), γ value and events per packet.

7.3 Corner Detector Experiment Results

The results of the event-based corner detection test (Figure 7.4) are shown in Figure 7.5. The γ filter operation is similar to the one shown in the previous section –γ decreases when the number of events generated increases in order to avoid overflowing–.

The homogenization of the algorithm temporal cost per event is even greater than in the previous case. It can be seen how it almost never exceeds the value 0.3 –i.e. 30% of the highest cost requiered–. These data are a good example of the adaptive nature of ASAP. Between $t = 50\,s$ and $t = 110\,s$ the number of generated events is not very high –the grid has not yet appeared in the field of view– so the number of discarded events and the number of events per packet will be low. In contrast, from $t = 110\,s$ the grid is seen by the camera and a lot of events are generated. The γ filter starts to discard more events and the number of events per packet increases.

7.4 Conclusions

The analysis of the results of the different experiments makes it possible to verify that ASAP achieves the expected operation. Both the event filtering and the event packing meet expectations, achieving high responsiveness with no processing saturation.

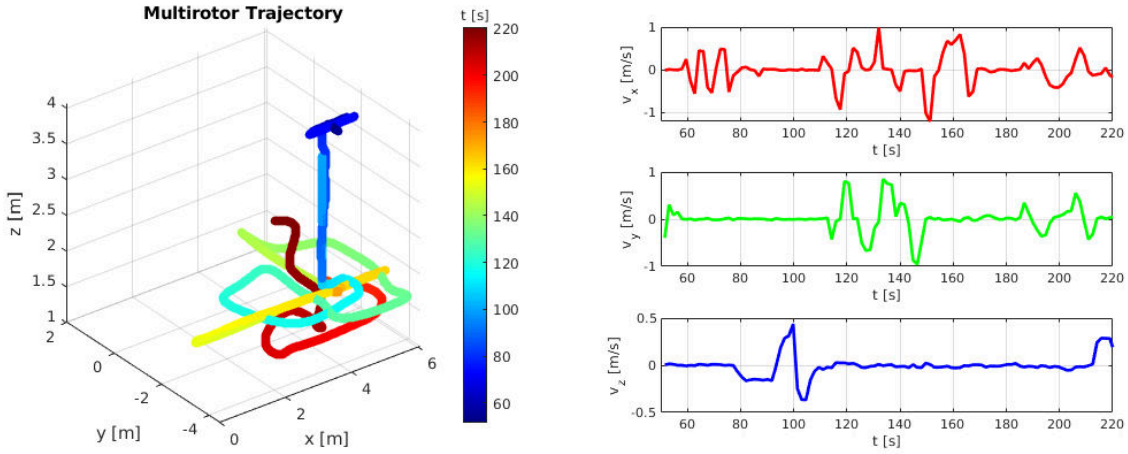

Figure 7.4 Multirotor trajectory during experiment. Position (left) and velocity (right) over time.

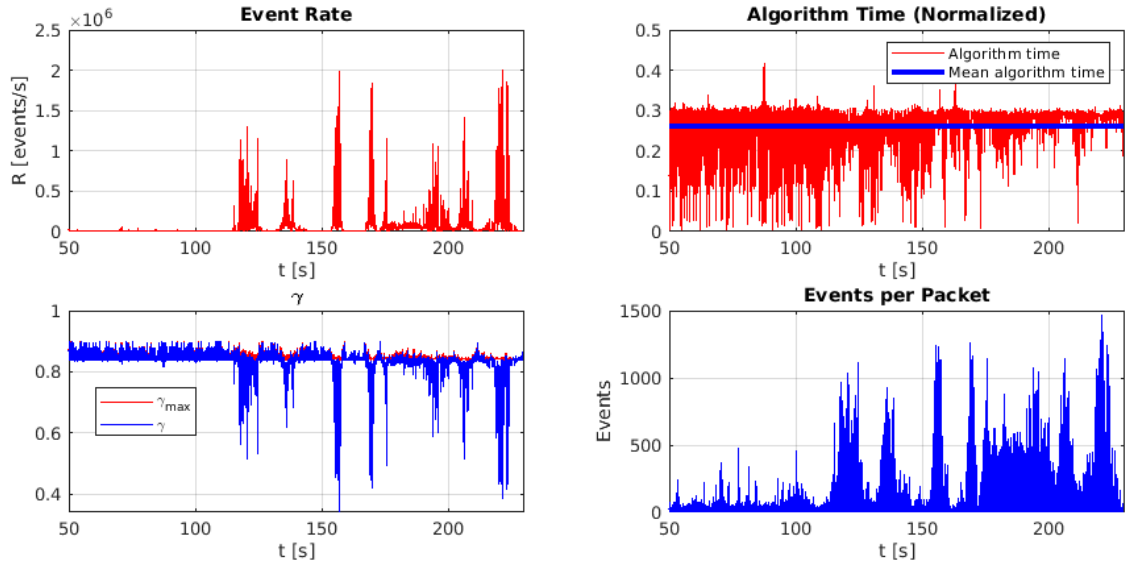

Figure 7.5 Event-based corner detection experiment results. From top-left: event rate, normalized algorithm temporal cost (from 0 to 1), γ value and events per packet.

The adaptive performance of this method is very important for asynchronous event processing. Therefore, it can be stated that ASAP is a desirable solution when handling the events stream for event-based algorithms.

These tests have also been used to check the ease of software integration. Although two different author algorithms with differing code structures have been used, connecting them to ASAP has not been a problem.

8 Conclusions and Future Work

This chapter presents the general conclusions drawn during the development of this project. This is followed by some new avenues for future research.

8.1 Conclusions

8.1.1 The Event Camera Revolution

Event cameras are a revolutionary technology that offers multiple advantages over traditional cameras. Their features make them a suitable option for robotic applications in scenarios inaccessible to other perception sensors.

The growing research and commercial interest reflected by different fields in the literature –e.g., neuromorphic systems, computer vision or robotics– is an indicator of the current expansion of these new sensors (see Figure 8.1).

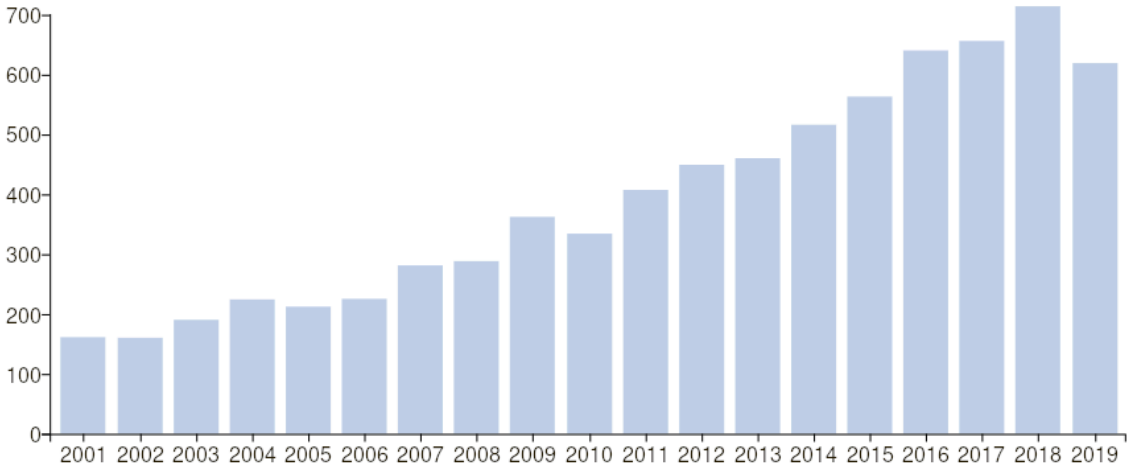

Figure 8.1 Number of papers per publication year for *event camera* topic. Graph extracted from Web of Science [49].

8.1.2 Event Processing

One of the biggest challenges of event-based research is the optimization of the number of events processed in each iteration by the vision algorithm. This choice will define the event-based perception system responsiveness

The two main approaches, event-by-event processing and event-packet processing, have advantages and disadvantages. The first approach achieves low latency, but can be overloaded during high-speed movements –where millions of events are generated– when running on a CPU. The second one can achieve a real-time operation with no need for a GPU, but it is not possible to reach a microsecond resolution. This project defends that an efficient solution would be a trade-off between both approaches.

8.1.3 ASAP Contribution to Event-Based Research

This document has presented ASAP, an adaptive scheme for dynamic event packing, specially designed for a fast and efficient event processing. This scheme responds to the paradigm shift that asynchronous event-based sensors has brought about in recent years.

There are not many event-based algorithms available nowadays compared to frame-based algorithms. However, the evolution of neuromorphic retinas tends to the appearance of more and more event-by-event algorithms. This is a logical evolution, since this type of processing allows to exploit all the advantages of event cameras.

Event processing requires a very high computational capacity, due to the high temporal resolution of event sensors. Advances in computer technology and computer science provide systems with increasing computing power (see Figure 8.2). However, although onboard computers also increase their processing power over the years, they still generally do not meet the restrictions imposed by the time resolution of event cameras.

8.1.4 ASAP in the Context of ERC Advanced Grant GRIFFIN Project

One of the biggest limitations of the current GRIFFIN winged robot prototypes is their payload. For this reason, the onboard electronics must be as light as possible, resulting in a limitation in processing capacity.

This is where ASAP becomes relevant, since it allows to take full advantage of the ornithopter's processing system. An efficient vision system is indispensable, as it will serve as a basis for manoeuvres planning and control.

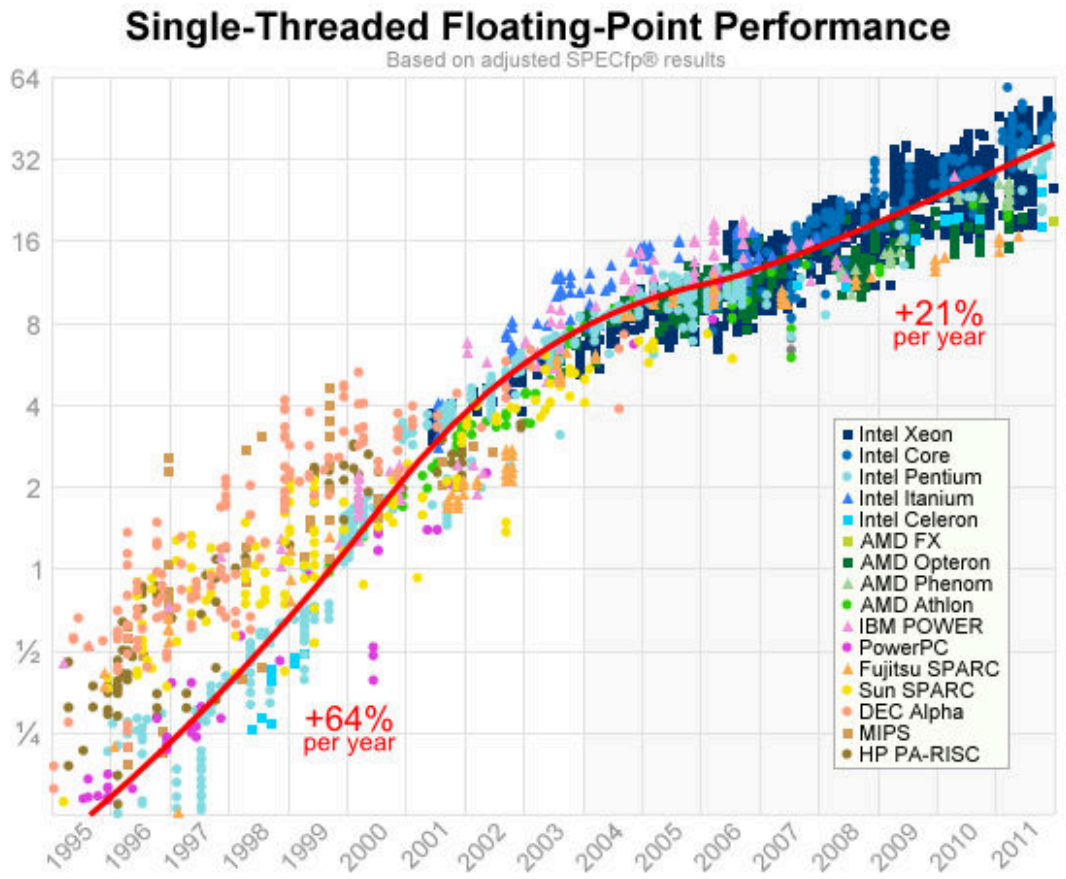

Figure 8.2 Floating-Point performance over time. Ordinate axis adjusted according to SPEC CPU2006 benchmark. Figure extracted from Preshing on Programming [50], where Standard Performance Evaluation Corporation (SPEC) [72] data were used.

8.2 Future Work

This section will describe some possible lines of future work to focus. Some are currently being explored, while others are ideas raised during the development of this project that may serve to improve some features of the method described.

8.2.1 Proportional Gain Adjustment Method

The value of K_p used by the event packer is manually adjusted to set the desired responsiveness level –considering that the faster the response, the greater the workload for the algorithm–.

A possible future development is the design of an analytical method that, given a desired system behaviour, allows to obtain an appropriate value of the proportional gain.

8.2.2 Packing Adaptation Techniques

The design of ASAP is based on some discrete systems theory techniques. The choice of a proportional gain as the only control parameter is due to the fact that the experimental results obtained verify the correct operation of the method.

However, other alternatives could be explored. For example, the inclusion of a derivative gain that improves the transient behaviour or tools from adaptive control.

Analysing the obtained experimental results, it has been verified the correct adaptation of ASAP, both to the environment and to the requirements of the event-based vision algorithm. Therefore, this should not be focused as a main task, since the improvement may not be significant.

8.2.3 Low-Level ASAP Version

The development of a low-level ASAP version for devices with very low processing capacity is currently underway. This version is being implemented in C without using ROS, as it is intended to reduce its computational complexity as much as possible.

Since it is not under ROS framework, a new communication structure must be designed to handle event traffic. This structure must be efficient, robust and low-latency.

The main disadvantage is the loss of portability, since ROS is a widely used middleware in robotics research. However, it will allow access to less power-consuming and cheaper technology.

8.2.4 Implementation on Board a Winged Aerial Robot

One of the final objectives of ASAP is to allow the processing of events on board the H2020 GRIFFIN project ornithopters. Event-based vision is necessary to avoid motion blur caused by flapping vibrations and for being able to perform manoeuvrers in low-light scenarios.

Payload restrictions mean that onboard electronics should be kept to a minimum. This implies that the processing capacity will not be as high as in other robotic vehicles, being necessary an efficient management of the generated events.

8.2.5 Future Publications and Open Source Version

All the work generated will result in a paper for a special issue of IEEE International Conference on Robotics and Automation (ICRA) 2020 Workshop on Unconventional Sensors in Robotics, where a preliminary version of ASAP has already been accepted [73].

This publication will be accompanied with the release of ASAP as open source for free access by the event-based perception community. The focus should be on

facilitating the integration of different algorithms through a multi-platform interface that allows them to be connected.

9 Conclusiones

Esta es una adaptación al español de las conclusiones del Capítulo 8.

9.1 La revolución de las cámaras de eventos

Las cámaras de eventos son una tecnología revolucionaria que ofrece múltiples ventajas en comparación con las cámaras tradicionales. Sus características las convierten en una opción adecuada para aplicaciones en el campo de la robótica, especialmente en escenarios inaccesibles por otros sensores de percepción.

El creciente interés investigativo y comercial reflejado en la literatura de distintos ámbitos (por ejemplo, sistemas neuromórficos, visión artificial o robótica) es un indicador de la actual expansión de estos nuevos sensores (véase la Figura 9.1).

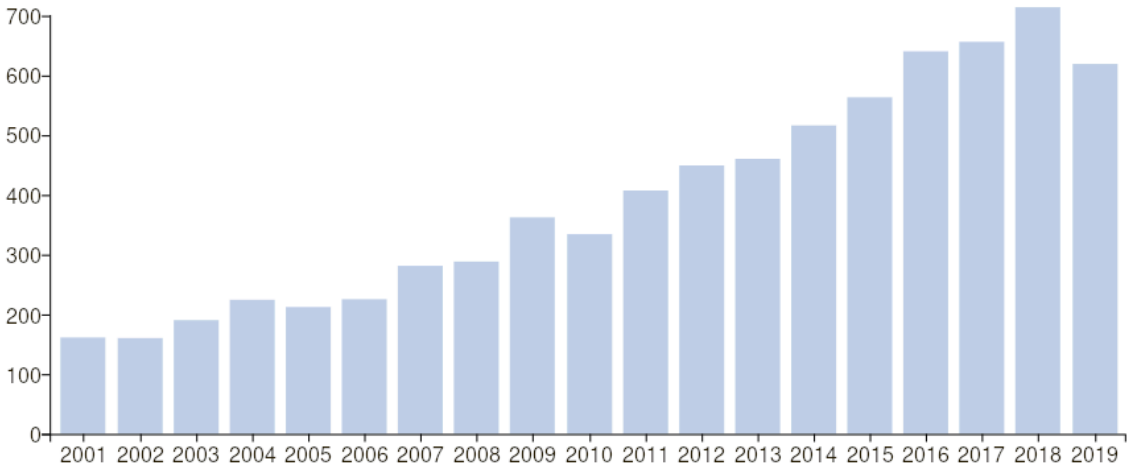

Figure 9.1 Número de artículos por año de publicación para el tema *cámara de eventos*. Gráfica obtenida de *Web of Science* [49].

9.2 Procesamiento de eventos

Uno de los mayores desafíos en el ámbito de las retinas artificiales es la optimización del número de eventos a procesar en cada iteración del algoritmo de visión. Esta elección definirá la capacidad de respuesta del sistema de percepción basado en eventos.

Los dos enfoques principales, procesamiento evento a evento y procesamiento de paquetes de eventos, poseen ventajas y desventajas. El primero logra una baja latencia, pero puede provocar una saturación con movimientos de gran velocidad (momentos en los que se generan millones de eventos) al ejecutarse en CPU. El segundo puede lograr un funcionamiento en tiempo real sin necesidad de ejecución en GPU, pero no es posible alcanzar una resolución del orden de microsegundos. Este proyecto defiende un compromiso entre ambos enfoques como una solución eficiente.

9.3 Contribución de ASAP a la percepción basada en eventos

Este documento ha presentado ASAP, un esquema adaptativo para el empaquetado dinámico de eventos, especialmente diseñado para un procesamiento de eventos rápido y eficiente. Este método responde al cambio de paradigma que los sensores asíncronos de eventos han supuesto en los últimos años.

No existe actualmente un gran número de algoritmos basados en eventos en comparación con el número de algoritmos basados en imágenes. Sin embargo, la evolución de las retinas neuromórficas tiende a la aparición de cada vez más algoritmos con procesamiento evento a evento. Esta es una evolución lógica, ya que este tipo de procesamiento permite explotar al máximo las ventajas de las cámaras de eventos.

El procesamiento de eventos requiere una capacidad computacional muy elevada, dada la alta resolución temporal de estos sensores. Los avances en las tecnologías de la computación y la información proporcionan sistemas con una potencia de cálculo cada vez mayor (véase la Figura 9.2). Sin embargo, aunque los ordenadores de abordo también ven incrementada su potencia de cálculo con los años, todavía no son capaces, por lo general, de satisfacer las restricciones impuestas por la resolución temporal de las cámaras de eventos.

9.4 ASAP bajo el marco del proyecto ERC Advanced Grant GRIFFIN

Una de las mayores limitaciones de los prototipos de robots de ala batiente en GRIFFIN es su carga de pago. Por ello, la electrónica de abordo debe ser lo más ligera posible, lo que implica una limitación en su capacidad de procesamiento.

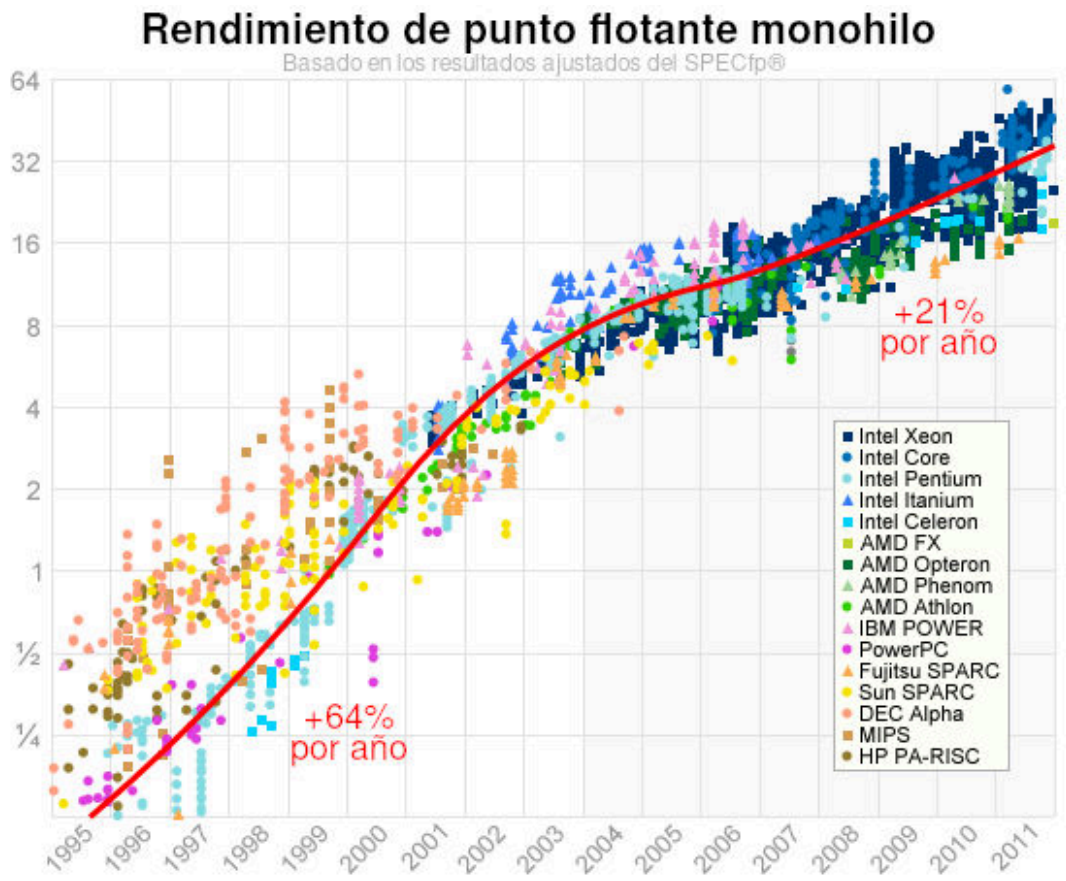

Figure 9.2 Rendimiento de la operación de punto flotante a lo largo del tiempo. Eje de ordenadas ajustado de acuerdo a la prueba de evaluación SPEC CPU2006. Figura extraída de *Preshing on Programming* [50], donde se utilizaron datos de la *Standard Performance Evaluation Corporation (SPEC)* [72]. Versión traducida al español.

Es en este punto donde ASAP cobra importancia, pues permite aprovechar al máximo el sistema de procesamiento del ornitóptero. Un sistema de visión eficiente es indispensable, ya que servirá de base para la planificación de maniobras y para el control de la plataforma.

Appendix A
ASAP Implementation

This appendix presents the ASAP algorithm for its implementation. It also shows the structure and communications of the ROS nodes during its execution.

A.1 ASAP Algorithm

The complete implementation of ASAP is shown in Algorithm 8. The different parameters used have been described throughout the chapters of this document.

A.2 ROS Node Structure

Figure A.1 presents the ROS node structure during the ASAP running. Besides events –*/asap/events*–, DAVIS346 device also outputs grayscale images –*/asap/images*– and data from an IMU integrated into the camera –*/asap/imu*–.

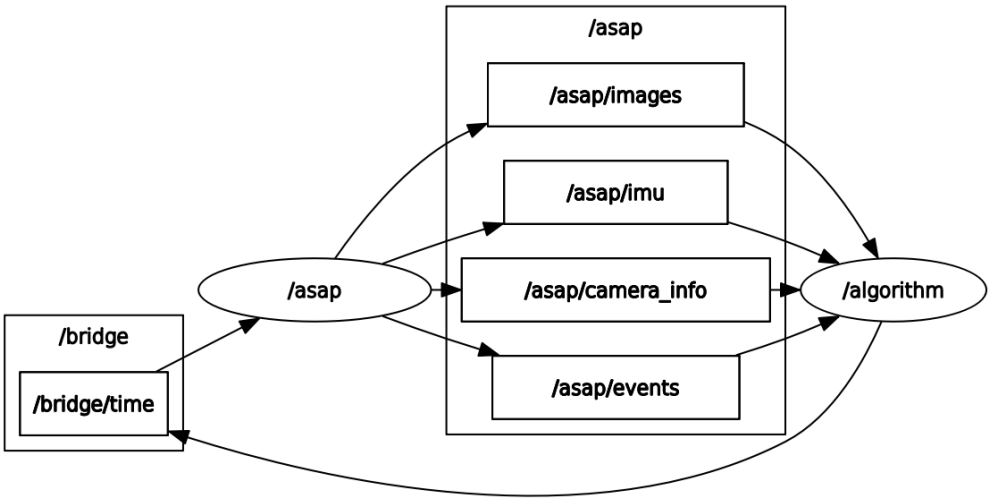

Figure A.1 ROS node and topic network.

Algorithm 8: ASAP

Input : e, $\Delta\bar{t}$
Output : E or no output
Parameters : N_R, γ_{min}, γ_{max}^{BOT}, γ_{max}^{TOP}, S_{min}, S_{max}, K_p, $\Delta\bar{t}_{ref}$
Initialization : E $\leftarrow$ emptyList(), $R_{list} \leftarrow$ emptyList(), R $\leftarrow$ 0, $\gamma \leftarrow \gamma_{max}^{TOP}$

$R_{list} \leftarrow$ insert(R_{list}, e)
if length(R_{list}) > N_R **then**
  $R_{list} \leftarrow$ removeOldestItem(R_{list})
  $r_n \leftarrow$ getNewestItem(R_{list})
  $r_o \leftarrow$ getOldestItem(R_{list})
  $t_r \leftarrow$ getTime(r_n) - getTime(r_o)
  R $\leftarrow$ N_R / t_r
end
r $\leftarrow$ generateRandomValue(0,1)
if r < γ **then**
  E $\leftarrow$ insert(E, e)
  S $\leftarrow$ S + $K_p \cdot (\Delta\bar{t}$ - $\Delta\bar{t}_{ref})$
  if S > S_{max} **then**
    S $\leftarrow$ S_{max}
  else if S < S_{min} **then**
    S $\leftarrow$ S_{min}
  end
  $e_n \leftarrow$ getNewestItem(E)
  $e_o \leftarrow$ getOldestItem(E)
  s $\leftarrow$ getTime(e_n) - getTime(e_o)
  if s > S **then**
    sendToAlgorithm(E)
    E $\leftarrow$ emptyList()
  end
  $\gamma_{max} \leftarrow \gamma_{max}^{TOP} - (\gamma_{max}^{TOP} - \gamma_{max}^{BOT}) \cdot \Delta\bar{t}$
  if R > R_{max} **then**
    $R_{max} \leftarrow$ R
  end
  $\bar{R} \leftarrow$ R / R_{max}
  $\gamma = \gamma_{max} - (\gamma_{max} - \gamma_{min}) \cdot \bar{R}$
end

Bibliography

[1] Iñigo Alonso and Ana Murillo, *Ev-segnet: Semantic segmentation for event-based cameras*, 11 2018.

[2] Ignacio Alzugaray and Margarita Chli, *Ace: An efficient asynchronous corner tracker for event cameras*, 09 2018, pp. 653–661.

[3] _____, *Asynchronous corner detection and tracking for event cameras in real-time*, IEEE Robotics and Automation Letters **PP** (2018), 1–1.

[4] Mehdi Azadmehr and J. Abrahamsen, *A foveated aer imager chip*, International Symposium on Circuits and Systems, ISCAS 2005 **3** (2005).

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

[6] Francisco Barranco, Cornelia Fermüller, and Eduardo Ros, *Real-time clustering and multi-target tracking using event-based sensors*, 10 2018, pp. 5764–5769.

[7] Francisco Barranco, Ching Teo, Cornelia Fermüller, and Yiannis Aloimonos, *Contour detection and characterization for asynchronous event sensors*, 12 2015, pp. 486–494.

[8] Souptik Barua, Yoshitaka Miyatani, and Ashok Veeraraghavan, *Direct face detection and video reconstruction from event cameras*, 03 2016, pp. 1–9.

[9] Ryad Benosman, Sio-Hoi Ieng, Charles Clercq, Chiara Bartolozzi, and Mandyam Srinivasan, *Asynchronous frameless event-based optical flow*, Neural networks : the official journal of the International Neural Network Society **27** (2011), 32–7.

[10] Kwabena Boahen, *A burst-mode word-serial address-event link—i: Transmitter design*, Departmental Papers (BE) **51** (2004).

[11] Christian Brändli, Raphael Berner, Minhao Yang, S.-C Liu, and Tobi Delbruck, *A 240 × 180 130 db 3 µs latency global shutter spatiotemporal vision sensor*, Solid-State Circuits, IEEE Journal of **49** (2014), 2333–2341.

[12] Christian Brändli, Thomas Mantel, Marco Hutter, Mark Hoepflinger, Raphael Berner, Roland Siegwart, and Tobi Delbruck, *Adaptive pulsed laser line extraction for terrain reconstruction using a dynamic vision sensor*, Frontiers in neuroscience **7** (2013), 275.

[13] Luis Camuñas-Mesa, Teresa Serrano-Gotarredona, Sio-Hoi Ieng, Ryad Benosman, and Bernabé Linares-Barranco, *On the use of orientation filters for 3d reconstruction in event-driven stereo vision*, Frontiers in neuroscience **8** (2014), 48.

[14] Luis Camuñas-Mesa, Teresa Serrano-Gotarredona, Bernabé Linares-Barranco, Sio-Hoi Ieng, and Ryad Benosman, *Event-driven stereo vision with orientation filters*, 06 2014, pp. 257–260.

[15] João Carneiro, Sio-Hoi Ieng, Christoph Posch, and Ryad Benosman, *Asynchronous event-based 3d reconstruction from neuromorphic retinas*, Neural Networks **45** (2013).

[16] CelePixel, *http://www.celepixel.com/#/Home*, June 2020.

[17] Andrea Censi and Davide Scaramuzza, *Low-latency event-based visual odometry*, 06 2014.

[18] Xavier Clady, Sio-Hoi Ieng, and Ryad Benosman, *Asynchronous event-based corner detection and matching*, Neural Networks **in press** (2015).

[19] Eugenio Culurciello, Ralph Etienne-Cummings, and Kwabena Boahen, *A biomorphic digital image sensor*, Solid-State Circuits, IEEE Journal of **38** (2003), 281 – 294.

[20] Tobi Delbruck, Bernabé Linares-Barranco, Eugenio Culurciello, and Christoph Posch, *Activity-driven, event-based vision sensors*, 05 2010, pp. 2426–2429.

[21] Tobi Delbruck and Carver Mead, *Time-derivative adaptive silicon photoreceptor array*, Proceedings of SPIE - The International Society for Optical Engineering (1993).

[22] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Jorg Conradt, Kostas Daniilidis, and Davide Scaramuzza, *Event-based vision: A survey*, (2019).

[23] Guillermo Gallego, Christian Forster, Elias Mueggler, and Davide Scaramuzza, *Event-based camera pose tracking using a generative event model*, (2015).

[24] Rohan Ghosh, Tang Siyi, Mahdi Rasouli, N.v Thakor, and Sunil Kukreja, *Pose-invariant object recognition for event-based vision with slow-elm*, 09 2016, pp. 455–462.

[25] Arren Glover and Chiara Bartolozzi, *Event-driven ball detection and gaze fixation in clutter*, 10 2016, pp. 2203–2208.

[26] _____, *Robust visual tracking with a freely-moving event camera*, 09 2017, pp. 3769–3776.

[27] Arren Glover, Valentina Vasco, Massimiliano Iacono, and Chiara Bartolozzi, *The event-driven software library for yarp—with algorithms and icub applications*, Frontiers in Robotics and AI **4** (2018).

[28] Augusto Gómez Eguíluz, Juan Gómez, Julio L. Paneque, Pedro Grau, J. Ramiro Martinez-de Dios, and Anibal Ollero, *Towards flapping wing robot visual perception: Opportunities and challenges*, 11 2019.

[29] iniVation, *https:// inivation.com/*, June 2020.

[30] Insightness, *https:// www.insightness.com/*, June 2020.

[31] Hanme Kim, Ankur Handa, Ryad Benosman, Sio-Hoi Ieng, and Andrew Davison, *Simultaneous mosaicing and tracking with an event camera*, 01 2014, pp. 1–12.

[32] Hanme Kim, Stefan Leutenegger, and Andrew Davison, *Real-time 3d reconstruction and 6-dof tracking with an event camera*, vol. 9910, 10 2016, pp. 349–364.

[33] J. Kramer, *An on/off transient imager with event-driven, asynchronous readout*, 02 2002, pp. II–165.

[34] Beat Kueng, Elias Mueggler, Guillermo Gallego, and Davide Scaramuzza, *Low-latency visual odometry using event-based feature tracks*, 10 2016, pp. 16–23.

[35] GRVC Robotics Laboratory, *https:// grvc.us.es/*, June 2020.

[36] Hongmin Li, Guoqi Li, and Luping Shi, *Classification of spatiotemporal events based on random forest*, vol. 10023, 11 2016, pp. 138–148.

[37] Ruoxiang Li, Dianxi Shi, Yongjun Zhang, Kaiyue Li, and Ruihao Li, *Fa-harris: A fast and asynchronous corner detector for event cameras*, 09 2019.

[38] P Lichesteiner, C Posch, and T Delbruck, *A 128× 128 120 db 15µsec latency asynchronous temporal contrast vision sensor*, IEEE Jornal of Solid-State Circuits **43** (2008), no. 2, 566–576.

[39] Patrick Lichtsteiner, Tobi Delbruck, and J. Kramer, *Improved on/off temporally differentiating address-event imager*, 01 2005, pp. 211 – 214.

[40] Shih-Chii Liu, Tobi Delbruck, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas, *Event-based neuromorphic systems*, 12 2014.

[41] Misha Mahowald, *Vlsi analogs of neuronal visual processing: a synthesis of form and function*, Ph.D. thesis, California Institute of Technology Pasadena, 1992.

[42] Alexandre Marcireau, Sio-Hoi Ieng, and Ryad Benosman, *Sepia, tarsier, and chameleon: A modular c++ framework for event-based computer vision*, Frontiers in Neuroscience **13** (2020), 1338.

[43] Diederik Paul Moeys, Federico Corradi, Emmett Kerr, Philip Vance, Gautham Das, Dan Neil, Dermot Kerr, and Tobi Delbruck, *Steering a predator robot using a mixed frame/event-driven convolutional neural network*, 06 2016.

[44] Elias Mueggler, Chiara Bartolozzi, and Davide Scaramuzza, *Fast event-based corner detection*, 09 2017.

[45] Elias Mueggler, Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza, *Continuous-time visual-inertial odometry for event cameras*, IEEE Transactions on Robotics **PP** (2018), 1–16.

[46] Elias Mueggler, Guillermo Gallego, and Davide Scaramuzza, *Continuous-time trajectory estimation for event-based vision sensors*, 07 2015.

[47] Elias Mueggler, Basil Huber, and Davide Scaramuzza, *Event-based, 6-dof pose tracking for high-speed maneuvers*, 09 2014.

[48] Rajkumar Muthusamy, Abdulla Ayyad, Mohamad Halwani, Yahya Zweiri, Dongming Gan, and Lakmal Seneviratne, *Neuromorphic eye-in-hand visual servoing*, 04 2020.

[49] Clarivate Analytics. Web of Science, *https://clarivate.com/products/web-of-science/*, June 2020.

[50] Preshing on Programming. A Look Back at Single-Threaded CPU Performance, *https://preshing.com/20120208/a-look-back-at-single-threaded-cpu-performance/*, June 2020.

[51] Garrick Orchard, Ryad Benosman, Ralph Etienne-Cummings, and N.v Thakor, *A spiking neural network architecture for visual motion estimation*, 10 2013, pp. 298–301.

[52] Paul Park, Evgeny Soloveichik, Hyunsurk Ryu, Jun Kim, Chang Shin, Hyunku Lee, Weiheng Liu, Qiang Wang, Yohan Roh, Jeonghan Kim, and Yotam Ater, *Low-latency interactive sensing for machine vision*, 12 2019, pp. 10.6.1–10.6.4.

[53] Christoph Posch, *Bio-inspired vision*, Journal of Instrumentation **7** (2012).

[54] ———, *Bioinspired vision sensing*, pp. 11–28, 08 2015.

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

[56] Christoph Posch, Teresa Serrano-Gotarredona, Bernabé Linares-Barranco, and Tobi Delbruck, *Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output*, Proceedings of the IEEE **102** (2014), 1470–1484.

[57] Prophesse, *https:// www.prophesee.ai/*, June 2020.

[58] Fran Real, Arturo Torres-González, Pablo Ramón Soria, Jesús Capitán, and Anibal Ollero, *Unmanned aerial vehicle abstraction layer: An abstraction layer to operate unmanned aerial vehicles*, International Journal of Advanced Robotic Systems (2020), 1–10.

[59] Henri Rebecq, Guillermo Gallego, Elias Mueggler, and Davide Scaramuzza, *Emvs: Event-based multi-view stereo—3d reconstruction with an event camera in real-time*, International Journal of Computer Vision (2017).

[60] Henri Rebecq, Guillermo Gallego, and Davide Scaramuzza, *Emvs: Event-based multi-view stereo*, 09 2016.

[61] Henri Rebecq, Timo Horstschaefer, Guillermo Gallego, and Davide Scaramuzza, *Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real-time*, IEEE Robotics and Automation Letters **PP** (2016).

[62] Christian Reinbacher, Gottfried Munda, and Thomas Pock, *Real-time intensity-image reconstruction for event cameras using manifold regularisation*, International Journal of Computer Vision (2016).

[63] Juan Pablo Rodríguez-Gómez, Augusto Gómez Eguíluz, José Ramiro Martínez De-Dios, and Anibal Ollero, *Asynchronous event-based clustering and tracking for intrusion monitoring in uas*, 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020.

[64] Paul Rogister, Ryad Benosman, Sio-Hoi Ieng, Patrick Lichtsteiner, and Tobi Delbruck, *Asynchronous event-based binocular stereo matching*, IEEE transactions on neural networks and learning systems **23** (2012), 347–353.

[65] Antoni Rosinol, Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza, *Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high speed scenarios*, IEEE Robotics and Automation Letters **PP** (2018), 1–1.

[66] Pierre-François Rüedi, Pascal Heim, F. Kaess, Eric Grenet, Friedrich Heitger, Pierre-Yves Burgi, Stève Gyger, and Pascal Nussbaum, *A 128 × 128 pixel 120-db dynamic-range vision-sensor chip for image contrast and orientation extraction*, Solid-State Circuits, IEEE Journal of **38** (2004), 2325 – 2333.

[67] Samsung, *https:// www.samsung.com/*, June 2020.

[68] Cedric Scheerlinck, Nick Barnes, and Robert Mahony, *Continuous-time intensity estimation using event cameras*, pp. 308–324, 05 2019.

[69] Teresa Serrano-Gotarredona, Bernabé Linares-Barranco, Francesco Galluppi, L. Plana, and Steve Furber, *Convnets experiments on spinnaker*, 05 2015, pp. 2405–2408.

[70] Chen Shoushun and A. Bermak, *Arbitrated time-to-first spike cmos image sensor with on-chip histogram equalization*, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **15** (2007), 346 – 357.

[71] Bongki Son, Yunjae Suh, Sungho Kim, Heejae Jung, Jun-Seok Kim, Changwoo Shin, Keunju Park, Kiyeung Lee, Jinman Park, Jooyeon Woo, Yohan Roh, Hyunku Lee, Yibing Wang, Ilia Ovsiannikov, and Hyunsurk Ryu, *A 640×480 dynamic vision sensor with a 9µm pixel and 300meps address-event representation*, 02 2017, pp. 66–67.

[72] Standard Performance Evaluation Corporation (SPEC), *https://www.spec.org/*, June 2020.

[73] Raul Tapia, Augusto Gómez Eguíluz, J. Ramiro Martinez-de Dios, and Anibal Ollero, *Asap: Adaptive scheme for asynchronous processing of event-based vision algorithms*, 06 2020.

[74] CORDIS. GRIFFIN (*General compliant aerial Robotic manipulation system Integrating Fixed and Flapping wings to INcrease range and safety*) ERC Advanced Grant Project, *https://cordis.europa.eu/project/id/788247*, June 2020.

[75] GRIFFIN (*General compliant aerial Robotic manipulation system Integrating Fixed and Flapping wings to INcrease range and safety*) ERC Advanced Grant Project, *https://griffin-erc-advanced-grant.eu/*, June 2020.

[76] Valentina Vasco, Arren Glover, and Chiara Bartolozzi, *Fast event-based harris corner detection exploiting the advantages of event-driven cameras*, 10 2016, pp. 4144–4149.

[77] Samsung SmartThings Vision, *https://www.samsung.com/au/smart-home/smartthings-vision-u999/*, June 2020.

[78] David Weikersdorfer, Raoul Hoffmann, and Jorg Conradt, *Simultaneous localization and mapping for event-based vision systems*, vol. 7963, 09 2013, pp. 133–142.

[79] Kareem Zaghloul and Kwabena Boahen, *Optic nerve signals in a neuromorphic chip i: Outer and inner retina models*, IEEE transactions on bio-medical engineering **51** (2004), 657–66.

[80] ———, *Optic nerve signals in a neuromorphic chip ii: Testing and results*, IEEE transactions on bio-medical engineering **51** (2004), 667–75.

[81] Alex Zhu, Nikolay Atanasov, and Kostas Daniilidis, *Event-based feature tracking with probabilistic data association*, 05 2017, pp. 4465–4470.

[82] Dekai Zhu, Jinhu Dong, Zhongcong Xu, Canbo Ye, Yingbai Hu, Hang Su, Zhengfa Liu, and Guang Chen, *Neuromorphic visual odometry system for intelligent vehicle application with bio-inspired vision sensor*, 09 2019.

[83] Cho “Dan” and Taejae Lee, *A review of bioinspired vision sensors and their applications*, Sensors and Materials **27** (2015).

Index

Glossary

AER Address Event Representation. 22, 24, 35

APS Active Pixel Sensor. 4, 11

ASAP Adaptive Scheme for Asynchronous Processing. 2, 4, 5, 27, 28, 30, 31, 33, 35, 37, 38, 40, 43, 44, 47–49, 51, 52, 55, 57–60, 62, 64, 68, 69

ATIS Asynchronous Time-based Image Sensor. 11, 12, 20, 23

CCD Charged Coupled Devices. 7

CMOS Complementary Metal-Oxide Semiconductor. 7

DAVIS Dynamic and Active Pixel Vision Sensor. 4, 11, 12, 20, 23, 52

DVS Dynamic Vision Sensor. 4, 10–12, 20, 21, 23

ERC European Research Council. 2

GRIFFIN General compliant aerial Robotic manipulation system Integrating Fixed and Flapping wings to INcrease range and safety. 2, 62, 64, 68

GRVC Grupo de Robótica, Visión y Control (Robotics, Vision and Control Research Group). 2

ICRA International Conference on Robotics and Automation. 2, 64

IEEE Institute of Electrical and Electronics Engineers. 2, 64

ROS Robot Operating System. 4, 30, 52, 64