Color event-based computer vision : framework, prototype and applications

Alexandre Marcireau

SORBONNE UNIVERSITÉ

INSTITUT DE LA VISION PARIS

THÈSE DE DOCTORAT
DE SORBONNE UNIVERSITÉ

Spécialité : Ingénierie neuromorphique

École doctorale nº391: Sciences mécaniques, acoustique, électronique et robotique

réalisée

à l'Institut de la Vision - Équipe vision et calcul naturel

sous la direction de Ryad B. Benosman

présentée par

Alexandre MARCIREAU

pour obtenir le grade de :

DOCTEUR DE SORBONNE UNIVERSITÉ

Sujet de la thèse :

Vision par ordinateur évènementielle couleur : cadriciel, prototype et applications

soutenue le 23 septembre 2019

devant le jury composé de :

Pr.	Elisabetta Chicca	Rapporteur
D.R.	Stéphane Viollet	Rapporteur
Pr.	Bruno Gas	Examinateur
Pr.	Ryad B. Benosman	Directeur de thèse
Dr.	Siohoi Ieng	Co-encadrant
Dr.	Jean Livet	Invité

Vision par ordinateur évènementielle couleur : cadriciel, prototype et applications

Résumé : L'ingénierie neuromorphique aborde de manière bio-inspirée le design des capteurs et ordinateurs. Elle prône l'imitation du vivant à l'échelle du transistor, afin de rivaliser avec la robustesse et la faible consommation des systèmes biologiques. Les caméras évènementielles ont vu le jour dans ce cadre. Elles possèdent des pixels indépendants qui détectent de manière asynchrone les changements dans leur champ visuel. La sortie du capteur est une séquence d'évènements épars possédant une grande précision temporelle. Ces propriétés étant mal exploitées par les algorithmes usuels de vision par ordinateur, un nouveau paradigme encourageant de petits calculs à chaque évènement a été développé. Cette approche témoigne d'un potentiel à la fois pour la vision par ordinateur et en tant que modèle biologique.
Le traitement de la couleur par le cerveau est encore mal compris, et les algorithmes de vision par ordinateur couleur sont peu génériques et coûteux en calcul. Le paradigme évènementiel peut fournir des solutions originales à ce problème.
Cette thèse explore la vision par ordinateur évènementielle, afin de mieux comprendre notre système visuel et identifier des applications. Nous approchons le problème par la couleur, un aspect peu exploré des capteurs évènementiels. Nous présentons un cadriciel supportant les évènements couleur, ainsi que deux dispositifs expérimentaux l'utilisant : une caméra couleur évènementielle et un système pour la psychophysique visuelle destiné à l'étude du temps précis dans le cerveau. Nous considérons l'application du capteur couleur à la méthode de génie génétique *Brainbow*, et présentons un modèle mathématique de cette dernière.

Mots clés : Neuromorphique, évènementiel, vision par ordinateur, couleur, Brainbow

Color event-based computer vision: framework, prototype and applications

Abstract: Neuromorphic engineering is a bio-inspired approach to sensors and computers design. It aims to mimic biological systems down to the transistor level, to match their unparalleled robustness and power efficiency. In this context, event-based vision sensors have been developed. Unlike conventional cameras, they feature independent pixels which asynchronously generate an output upon detecting changes in their field of view. The overall output is a sparse sequence of events with high temporal precision. These properties are not leveraged by conventional computer vision algorithms, thus a new paradigm has been devised. It advocates short calculations performed on each event to mimic the brain, and shows promise both for computer vision and as a model of biological vision.
Color processing in the brain is not fully understood, and color computer vision tasks are handled by ad hoc and computationally expensive algorithms. The event-based paradigm may provide new and original answers to the problem.
This thesis explores event-based computer vision to improve our understanding of visual perception and identify potential applications. We approach the issue through color, a mostly unexplored aspect of event-based sensors. We introduce a framework supporting color events, as well as two experimental devices leveraging it: a three-chip event-based camera performing absolute color measurements, and a visual psychophysics setup to study the role of precise-timing in the brain. We explore the possibility to apply the color sensor to the genetic engineering *Brainbow* method, and present a new mathematical model for the latter.

Keywords : Neuromorphic, event-based, computer vision, color, Brainbow

Alexandre
Marcireau

12, rue du Moulin Joly
75011 Paris
France

+33 7 82 09 24 11

alexandre.marcireau
@gmail.com

Language skills

French: native
English: fluent
Spanish: basics

Programming

C, C++, Python
Javascript,
git

Software

SolidWorks, Blender
Professional video editing
softwares

PhD student at Sorbonne Universités

Education

2015-2019	**PhD student in Neuromorphic Engineering at Sorbonne Universités, under the supervision of R. B. Benosman** Development of a color event-based sensor, and its application to the *Brainbow* method
2011-2015	**Engineer student at Centrale Paris** Specialization in computer science, in one of France's leading engineering schools. **Studying:** computer science, physics, mathematics
2009-2011	**Classes préparatoires:** Preparatory mathematics and physics class for the competitive examination to the French « Grandes Ecoles » for scientific studies

Laboratory interships

May-August 2015	**Vision Institute - France (Paris)** Development of a bio-inspired color camera and simulation algorithms for the *Brainbow* method
March-July 2014	**CEA Cadarache - France (Bouches-du-Rhône)** Uranium dioxyde Raman spectrum simulation using empirical potentials
August-February 2014	**Penn State University - USA (Pennsylvanie)** Research assistant in the conception and prototyping of open-source, self-replicating 3D printers

Publications

Accepted

Marcireau, A., Ieng, S. H., Simon-Chane, C., & Benosman, R. B. (2018). Event-based color segmentation with a high dynamic range sensor. *Frontiers in neuroscience*, *12*, 135.

Submitted

Submitted in June 2019 in Frontiers in neuroscience - Marcireau, A. et al., Sepia, Tarsier and Chameleon: a modular C++ framework for event-based computer vision.

In preparation

F*all 2019* - Marcireau, A. et al., Modelling and simulation of Cre-lox recombinations using Markovian processes.

Fall 2019 - Marcireau, A. et al., Hibiscus, a 1440 Hz projector and recording device for high-framerate psychophysics.

Workshops

2019 **Telluride Neuromorphic Cognition Engineering Workshop**
Implementation of the STICK framework on Intel's Loihi chip, and contribution to an automated neuromorphic Fussball.

2018 **CapoCaccia Cognitive Neuromorphic Engineering Workshop**
Development of an artificial fovea for object recognition on the Pushbot, with a DVS128 sensor and the ROLLS chip.

Other work experience

July 2012 **Blue-collar internship (6 weeks) at Airbus helicopters - France (Bouches-du-Rhône)**
Manual teamwork on the NH-90 helicopter assembly line

2009-2013 **Summer camp counsellor**
Supervise and entertain children aged between 6 and 13, working in a team

Personal projects

2018-2019 **Development of the Nautinov website, with UPS and credit card payment integration**

2017-2018 **Development of the Origami application - a graph-based interface to Google Scholar**

2014-2015 **Development of a phone app to report broken bikes in a bicycle sharing system**

2012-2014 **Tutoring a class of twenty students in mathematics and physics**

2012-2013 **Design and building of an airship carrying a video camera**

Acknowledgements

First, I would like to express my gratitude to my supervisor Pr. Ryad B. Benosman, for not only sharing some of his many ideas with me, but giving me the means and time to pursue mine.

I would also like to thank my advisor Dr. Siohoi Ieng for his support and constructive feedback on my work. I am grateful to Dr. Jean Livet for sharing his knowledge on the *Brainbow* method, and for fascinating discussions on Cre-*lox* models.

My sincere thanks go to *les Loulous*, my many past and present teammates. These were four unforgettable years; I will miss our amazing debates and questionable liquor choices.

I want to thank *On craint dégun* for bringing a bit of that sweet southern sun to the grey Parisian skies, and having been around for all these years.

I thank the *Jean Groupe Club* for always being there to remind me that, sometimes, I talk a little too much.

Finally, I am profoundly grateful to my grandparents, parents, brother, sister and all the other members of my family for their unfailing love and support. Thank you.

Contents

Introduction

At the turn of the 17th century, a most peculiar idea was born: the abstract mathematics of ancient Greeks, "a detached intellectual subject for the connoisseur" (Burton, 2007 [1]), are the language of our physical universe. This idea is best expressed by Galileo's famous sentence "Philosophy is written in this grand book, the universe[...]. It is written in the language of mathematics, and its characters are triangles, circles and other geometric figures" (Galileo and Drake, 1957 [2]). Together with rigorous experimental validation, it brought forth a scientific revolution. Over the subsequent centuries, mathematics were used to describe, understand and predict an increasing number of natural phenomenons, and applied to increasingly complex machines.

In 1936, Alan Turing introduced a theoretical machine able to compute any sequence of mathematical operations [3], thus able to interpret and predict natural phenomenons - very much like human beings. His work on the possibility for machines to "compete with men in all purely intellectual fields" (Turing, 1950 [4]) paved the way for artificial intelligence. Since then, machines have beaten top humans at difficult intellectual tasks such as chess [5] and go [6]. Yet, tasks that come easy to humans have proven unexpectedly difficult for machines. In particular, having a machine solve vision tasks (locating oneself in an environment, segmenting and recognizing objects...), once thought to be achievable within a summer internship [7], remains a challenge more than half a century later.

Despite similar goals, and in some cases abilities, the human brain is very different from computers. The latter, embodiment of the Universal Turing Machine, perform deterministic computations with separated memory and processing units. In contrast, our neurons are responsible for both, and manipulate information stochastically [8]. Likewise, the sensory systems leveraged by the brain extract and encode information using strategies and computational structures fundamentally different from their artificial counterparts.

It has been hypothesized that these structural differences are responsible for the poor performance of computers in dealing with real-world sensory problems, compared to biological systems. The first research works to structure artificial computers like biological systems, down to the transistor level, date back to the 1980s [9]. They laid the foundations to *neurormorphic engineering*, a bio-inspired approach to the design of sensors and processors [10]. Notable advances include silicon retinas, silicon cochleas, olfactory sensors [11], and neural processors [12].

The *Dynamic Vision Sensor* (or DVS) [13] presented in 2008 opened the way for applications, as the first silicon retina competitive with conventional sensors in terms of latency, power consumption, dynamic range and ease of use. Unlike conventional artificial sensors, each pixel of a DVS is asynchronous and independent. Pixels generate an output - called event - when the luminance in their individual field of view changes beyond a

fixed threshold. The sensor's output is a sequence of spatially sparse events which do not carry absolute luminance information. However, each event is time-stamped with a precision in the order of a few hundred microseconds.

Conventional computer vision algorithms do not take full advantage of the output of silicon retinas. The DVS, as well as more recent sensors based on a similar principle, triggered the development of a new branch in computer vision, referred to as *event-based computer vision*. This approach advocates small calculations performed on each event, in order to leverage the high temporal resolution of the sensor's data [14] and be compatible with neuromorphic processors [15]. It aims to outperform conventional computer vision using simplifying assumptions supported by the fact that the sensor's temporal resolution is large compared to the scene's dynamics. For example, one can assume that objects move by at most one pixel between two samples, easing tracking calculations [16].

Looking at computer vision under a new paradigm may provide new and original answers to long-standing problems. One such problem is color: a generic approach to its many applications [17] has yet to be found, as illustrated by the multiple ad hoc approaches of pre-deep learning computer vision [18]. Convolutional neural networks yield a more generic solution to the problem, however they require massive amounts of data and computational power. Moreover, the role of color is not easy to extract from their opaque models [19].

Neuromorphic engineering takes inspiration from multiple fields, notably neuroscience, biology and psychophysics. In turn, advances in engineering enable new experiments in these fields. Thus, a focus on neuromorphic engineering applications to biology yields a virtuous circle benefiting both research topics. As regards algorithms design, concrete applications provide great experimental validation, and help define the solution scope.

This thesis aims to explore neuromorphic event-based computer vision, to improve our understanding of visual perception and identify possible applications of this new paradigm. We will approach the issue through color, an aspect of event-driven sensors that underwent little exploration. The absolute exposure measurements provided by the *Asynchronous Time-based Image Sensors* (ATIS) [20] will be leveraged to perform absolute event-based color measurements. We will present a software framework supporting events with arbitrary payloads to represent these measurements. An event-based color sensor will be assembled from three ATIS, and used hand-in-hand with a novel algorithm to solve segmentation problems. The framework will also be used to build a psychophysics setup designed to better understand the role and importance of temporal precision in biological visual systems. The application of the color sensor to biological samples generated by the genetic engineering method *Brainbow* [21] will be discussed, and a new mathematical model of the latter will be introduced.

Chapter I

State of the art

I.1 Color sensing

Primates' ability to discriminate colors is advantageous for survival [22]. They use it for long-range detection of edible food in forest environments. This ability is nowadays exploited by humans to efficiently communicate information: road signs, merchandising and maps are a few examples. Consequently, machine vision systems meant to interface with humans or mimic their vision often rely on colors [23]. Applications include, among others, traffic sign recognition [24], skin detection [25], visual saliency modeling [26] or vehicle color classification [27]. This section gives an overall view of human color perception, and compares it to state-of-the-art architectures for artificial color sensing.

I.1.1 Human color perception

Photons emitted or reflected by an object are characterized by their distribution of power over the electromagnetic spectrum (among others things). This distribution lies in an infinite-dimensional space. The eyes of many animals - including humans - feature several types of photoreceptors, with distinct spectral sensitivities. The excitation of a photoreceptor can be calculated as the product of the power distribution by the energy-based spectral sensitivity, integrated over the spectrum [28]. The original infinite-dimensional power distribution is projected on a finite basis: different spectra can have the same representation in the finite basis. This property, known as metamerism, has notable consequences in artificial color creation and sensing [29]. Some animals with a single photo-receptor type (notably cephalopods and color-blind primates) exhibit color perception. A. L. Stubbs et al. [30] describe a mechanism enabling color sensing using chromatic blurring.

The human retina contains three types of photoreceptor cells: cones, rods and photosensitive retinal ganglions. The latter participate in circadian patterns and pupil reactions [31], whereas cones and rods are responsible for high precision spatio-temporal light sensing. Color vision is mostly attributed to cone cells, which exist in three types: S, M and L. Figure I.1 shows the relative energy-based spectral sensitivity of macaque photoreceptors, indistinguishable from human ones [32].

The photoreceptor excitation that can be derived from the sensitivity curves, given an electromagnetic spectrum, is only the first step in color perception. Young–Helmholtz's

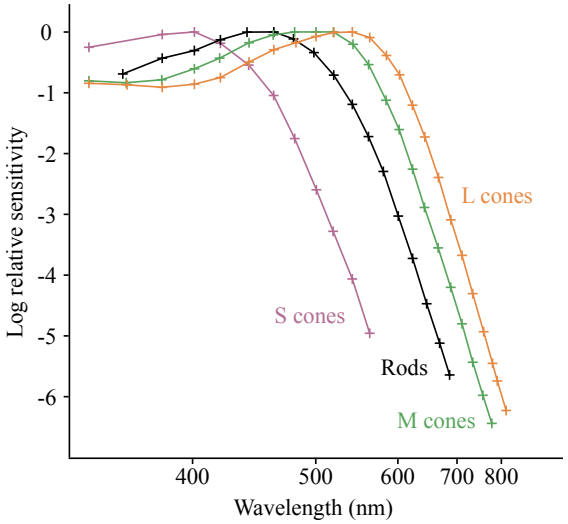


Fig. I.1 *Relative energy-based spectral sensitivity characterizes the relative excitation of photoreceptors to specific wavelengths. The curves shown in this figure were measured on a macaque. Indistinguishable results are obtained with human subjects [32]. Our photoreceptors are sensitive to overlapping ranges, notably the M and L cones. The blue, green and red colors were not used to represent cones on purpose, to avoid the frequent confusion between physical sensing and color perception. The x axis is linear in wavenumber, hence the uneven distribution of wavelength ticks. From E. R. Kandell et al. [8]*

three-receptor theory (1850) is one of the first attempts to describe color perception as a physical mechanism, and correctly postulated the existence of cone cells. However, it wrongfully assumed that color perception resulted from a linear combination of the cells activity. Hering's opponent colors theory (1892) described a competing mechanism, based on four colors (yellow, red, blue and green) arranged in mutually excluding pairs, and two achromatic colors (black and white). Though experiments proved this theory wrong [33], or at least incomplete, it remains critical as it brought forth the idea that color perception is not directly inferred from the photoreceptors' activity, but emerges further up in the visual pathway. The three-receptor and opponent theories were reconciled by Schödinger (1925), as two complementary stages of color perception, and further developed throughout the 20th century.

Color processing starts in the retina, where cone-opponent retinal ganglion cells combine the output of neighboring photoreceptor [34]. There are several types of ganglions, each performing a specific combination which takes into account the cone cells types and their relative spatial position. The ganglions' axons project to the lateral geniculate nucleus, or LGN, which in turn projects to the visual cortex. The role of the LGN in color processing, beyond relaying information from the eyes to the visual cortex, is not fully understood [35]. Despite advances in neuroscience, several open questions remain on the matter of color processing - and ultimately perception - in the visual cortex. Notably, there is no consensus on the extent to which color and visual shapes are treated separately, though the evidence accumulated in recent experiments tips the scale towards an

Fig. I.2 *Spatial context plays a fundamental role in the way we perceive colors. In this example, the words sepia and tarsier are written with the same color, as emphasized by the connector between their i letters. Likewise, cham and eleon are written with the same shade of orange. Adapted from S. K. Shevell et al. [39]*

inextricable link between color and form [36, 37].

Land's experiments, notably his 1955 demonstrations at the Annual Meeting of the Optical Society of America, showed the importance of spatial adaptation in color perception. An example is illustrated figure I.2. These experiments highlighted the shortcomings of existing theories, and paved the way for Retinex algorithms [38], which aim to mimic human color perception by modelling the retina, the LGN and the visual cortex.

Most artificial color sensors are based on Young–Helmholtz's three-receptor theory, with little on-chip processing beyond demosaicing. This limitation is not surprising, given that these sensors are meant to capture data for human consumption rather than to mimic biological systems. Multiple methods to separate color channels were successfully implemented, and are illustrated in the next section. The color transforms found in the opponent or Retinex theories, which have applications in color segmentation, are generally performed outside the sensor (see § I.4).

I.1.2 Artificial color sensing

Color sensors were originally developed for photography: their data was not meant to be analyzed by a machine, but to be displayed for human consumption. Given our trichromatic vision, it is enough for an artificial sensor to capture light in the same manner our cones cells would, by projecting the electromagnetic spectrum on a three-dimensional basis.

Screens and light projectors simply needs to emit a superposition of the three captured colors, a process known as additive mixing. Since this three-channel representation of color and the original source are metamers, they are identical to the human eye. Physical substances such as ink do not emit light, but reflect specific wavelengths: red ink absorbs everything but red light. Thus, mixing red and blue ink results in a black material absorbing every wavelength. In order to show arbitrary colors to the human eye, the three colors yellow, magenta and cyan ink must be used instead. They reflect every

wavelength but those most visible to the S, M and L cones (respectively). A combination of these colors is called subtractive mixing.

There are several methods to build artificial photoreceptors with spectral sensitivities similar to our cones. C. Wootton [40] lists three families of methods: multi-pass, filter arrays and beam splitting. The multi-pass approach was used by the Kinemacolor, which relied on a spinning wheel with red and green filters and a single strip [41]. Beam splitting could be found in Technicolor cameras [42], and is still used by 3 CCD chips. The latter rely on dichroic filters arranged in a prism configuration. Filter arrays, also known as Bayer filters, consist in a mosaic of blue, green and red filters in front of a single sensor. Bayer filters come in many variants [43]. They are the most widespread technology for building color sensors, for they are less bulky than 3 CCD cameras, and do not suffer from alignment issues. However, they require demosaicing algorithms [44] and create color artifacts. Color co-site sampling uses both multi-pass and an array of filters [45], circumventing Bayer filters limits at the expense of frame-rate. The Foveon sensor [46] combines the benefits of Bayer filters and 3 CCD cameras, thanks to stacked blue, green and red photoreceptors. The three approaches co-exist today, with on-going investigations to characterize the benefits - and possible drawbacks - of Foven sensors over Bayer filters [47]. Figure I.3 illustrates the different color sensor architectures.

Unlike usual color cameras, multispectral sensors do not try to mimic the three human cones. Instead, they capture multiple spectral bands (between three and several hundred) [48, 49] as to increase the amount and quality of data gathered from the scene. Notable applications include space imaging [50] and biology imaging [51]. Multispectral cameras use the same methods as RGB cameras to separate color components: rotating wheels [52], Bayer filters [53] or Foveon pixels [54].

In order to be represented mathematically or numerically, colors are expressed in three-dimensional spaces. Color spaces can be organized in three categories: device-oriented, user-oriented and device-independent [55]. Device-oriented color spaces are used to communicate with sensors, printers or displays. For example, the RGB color space directly encodes the output of color sensors, the CMY color space represents ink proportions for a printer, and YUV is used for TV broadcasting. They come in many linear and non-linear variants to support devices with higher dynamic ranges or color precision. User-oriented color spaces, such as HSV, are polar coordinate representations of the RGB space. The conversion from RGB to HSV can be expressed as a sequence of simple geometric transformations. These color spaces were designed for use in color selection tools: specifying a color using HSV values is more intuitive to humans than specifying it in RGB space [56]. Device-independent spaces, such as CIEL*a*b*, aim for perceptual uniformity: they are designed so that the Euclidean distance is a good estimate of the perceptual difference between colors. The transformation from RGB to CIEL*a*b* can be seen as a simple-to-compute model of human color vision. Perceptually uniform color spaces have many applications in computer vision. Figure I.4 shows the difference between Euclidean and perceptual distance in RGB and CIE L*a*b* spaces. Device-independent spaces can be represented in polar coordinates as well [57]. The relatively recent CIECAM02 color space, which takes into account stimulus' surroundings, is a more plausible model of the human visual cortex than CIE L*a*b* [58].

A new kind of artificial sensors, called silicon retinas, emerged in the 1990s. The next section outlines a brief history of these sensors, and shows how they lead to event-based

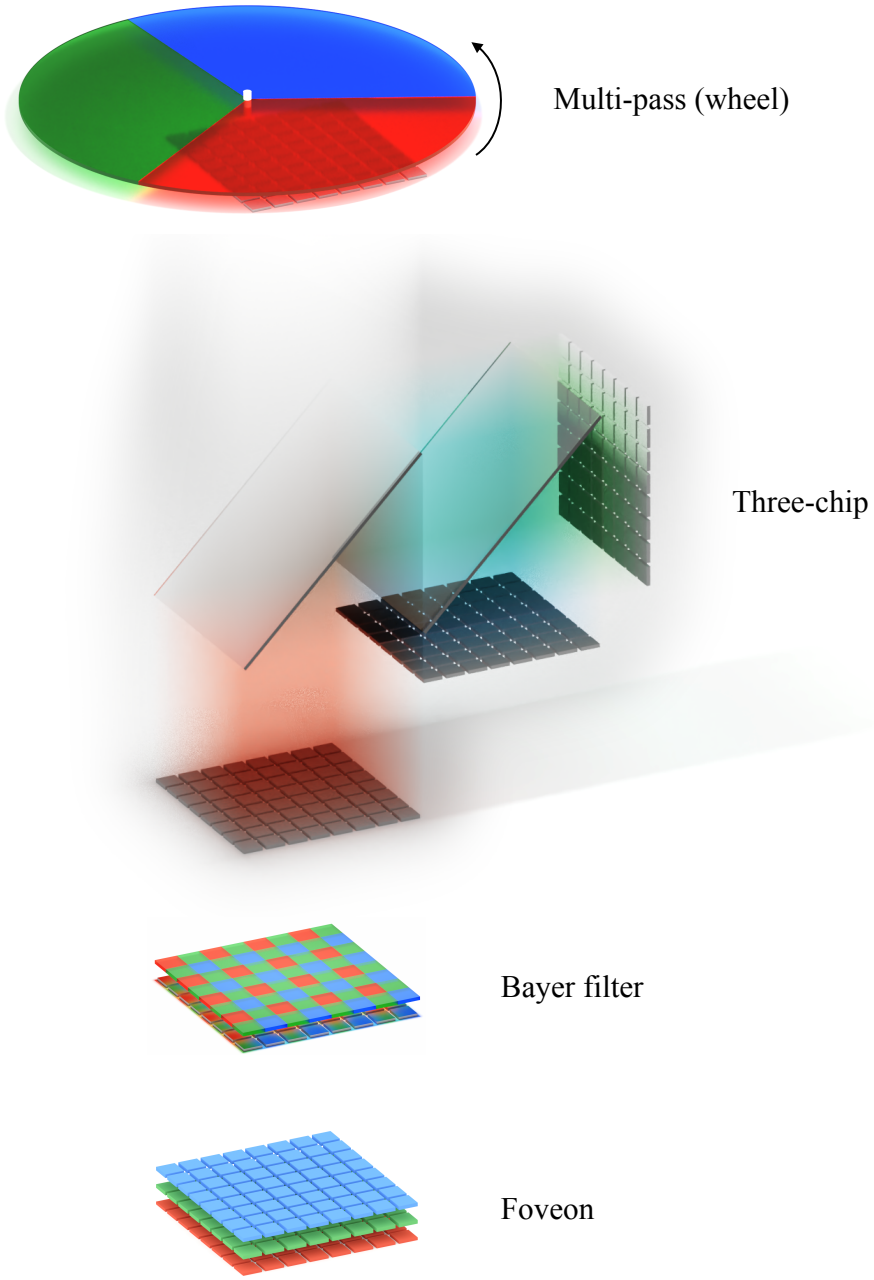


Fig. I.3 *In order to generate data compatible with human vision, color sensors split light into three components. Existing architectures either use a single chip and alternate samples over time (mutli-pass) or space (Bayer filter), or use multiple chips. Foveon sensors use stacked arrays of pixels, with one layer per color component. Multispectral sensors rely on the same techniques, however they use more, or different, base colors.*

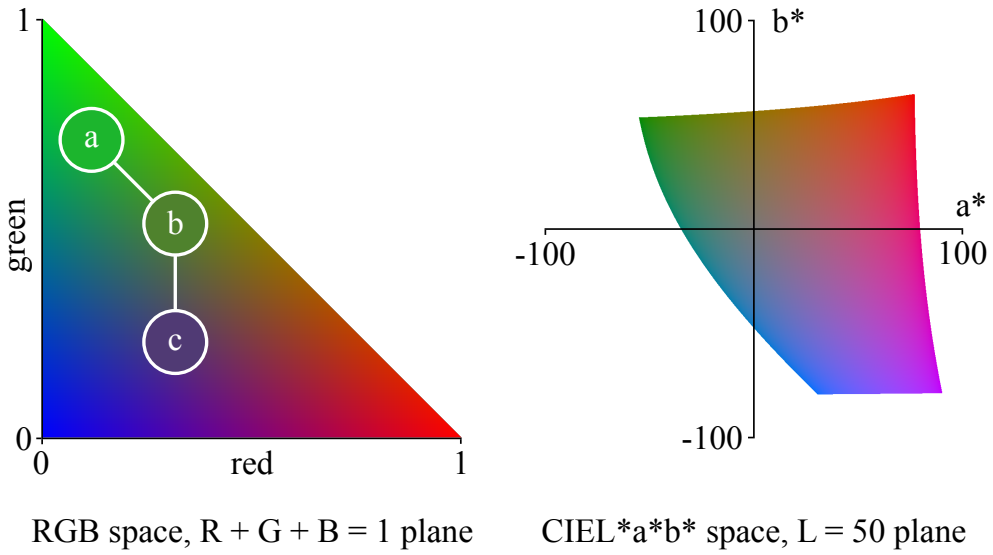


Fig. I.4 *The RGB space (left) is not perceptually uniform. A given Euclidean distance between two colors can correspond to visually similar colors (a and b) or very different ones (b and c). The CIEL*a*b* color space (right) is designed to be perceptually uniform, which simplifies the design of many computer vision algorithms. The conversion from RGB to CIEL*a*b* requires highly non-linear transforms which combine the original color channels.*

computer vision.

I.2 From the silicon retina to event-based sensors

I.2.1 History

The pixels of conventional digital cameras carry out a simple operation: they integrate light over fixed time intervals and convert the result to a digital value. Such pixels are generally implemented using either a charge-coupled device or a transistor [59]. Light integration is performed by all the pixels at the same time (global shutter) or sequentially (rolling shutter). In both cases, the integration duration is the same for every pixel. The sensor output is a spatially dense array of digital values - or frame - representing the average luminance over the time of integration.

By contrast, the human retina contains complex circuits performing analog operations on visual data [60]. This inspired the silicon retina, devised in 1989 by M. A. Mahowald and C. Mead [61]. It contains three major components mimicking biological retinas: a logarithmic photoreceptor, a local spatio-temporal averaging circuit and a spatial contrast amplification circuit. The circuits' calculations are performed directly on the current output of the photoreceptor, without any conversion. The chip's output is analog, with a sequential read-out similar to a rolling shutter. Subsequent sensors expanded on this principle, with dedicated circuits for edge detection, motion detection [62, 63], velocity measurement [64] or improved dynamic range [65]. Since these sensors were meant to be

integrated in fully neuromorphic hardware assemblies [66], with asynchronous and analog communications between components, little focus was given to digital conversion and compatibility with conventional computers.

Building fully neuromorphic systems is difficult, because one must simultaneously invent massively parallel analog hardware and novel algorithms [67]. Both are challenging: the methods resulting from years of research in digital hardware design do not necessarily apply to analog design, and the mechanisms governing the brain, source of inspiration for neuromorphic algorithms, are not fully understood yet. In order to separate the two problems, the research community devised ways to convert the output of silicon retinas to digital data. Thus, the matter of algorithms can be tackled with conventional, well understood computers, whereas the camera performs analog pre-processing. T. Delbrück et al. [68, 69] list several major contributions that paved the way for neuromorphic sensors compatible with digital computers. The introduction of *Address-Event Representation* [70] enabled transmission of precise timing information between chips (even over packet-switched networks, or buses without strong temporal constrains), and became a de facto standard for neuromorphic hardware. K. Boahen et al. [71] added the artificial equivalent of biological parallel pathways to silicon retinas, resulting in four distinct output channels: *increasing*, *on*, *decreasing* and *off*. The sensors introduced between 2003 and 2006 [72, 11] overcame the performance limits and mismatch issues of previous silicon retinas, and lead to the *Dynamic Vision Sensor* (or DVS) [13].

The DVS is an event-based sensor: upon meeting a specific condition, its pixels emit an output transmitted to the computer. The DVS implements the most widespread type of calculation among event-based sensors, namely brightness change detection. The pixel's photodiode output is continuously monitored to detect significant variations. When the logarithmic luminance changes beyond a fixed threshold, the pixel sends an event to the computer. This event bundles spatial and temporal information, as well as a boolean polarity encoding whether the significant change corresponds to an increase or decrease in brightness. Other sensors implement this behavior: the *color DVS* [73], the *Asynchronous Time-based Image Sensor* (or ATIS) [74] and the *Dynamic and Active-pixel Vision Sensor* (or DAVIS) [75]. They are still under active development, with improved versions featuring lower latency [76], higher sensitivity [77, 78], higher dynamic range [79], or more pixels [80]. Event-based cameras output their events in the order they are produced, resulting in a spatially sparse sequence with sub-millisecond precision, whereas conventional cameras output spatially dense frames. Figure I.5 highlights the difference between a sequence of frames and a stream of polarity events recorded from the same scene.

The cDVS and ATIS differ from the DVS by their extended pixel circuits generating a second type of polarity events, besides change detection. The polarity bit of the second event type encodes another visual information. The cDVS triggers such events on wavelength changes, whereas the ATIS encodes absolute exposure measurements in the time difference between them. The DAVIS is a hybrid sensor: it features both a DVS-like circuit and a light integration circuit. The latter produces frames similar to those generated by a conventional sensor. [81] present another event-based sensor, the *CeleX*, with a behavior similar to that of a DVS: events are triggered by brightness changes. However, output events include an absolute exposure measurement encoded on 9 bits instead of a binary polarity.

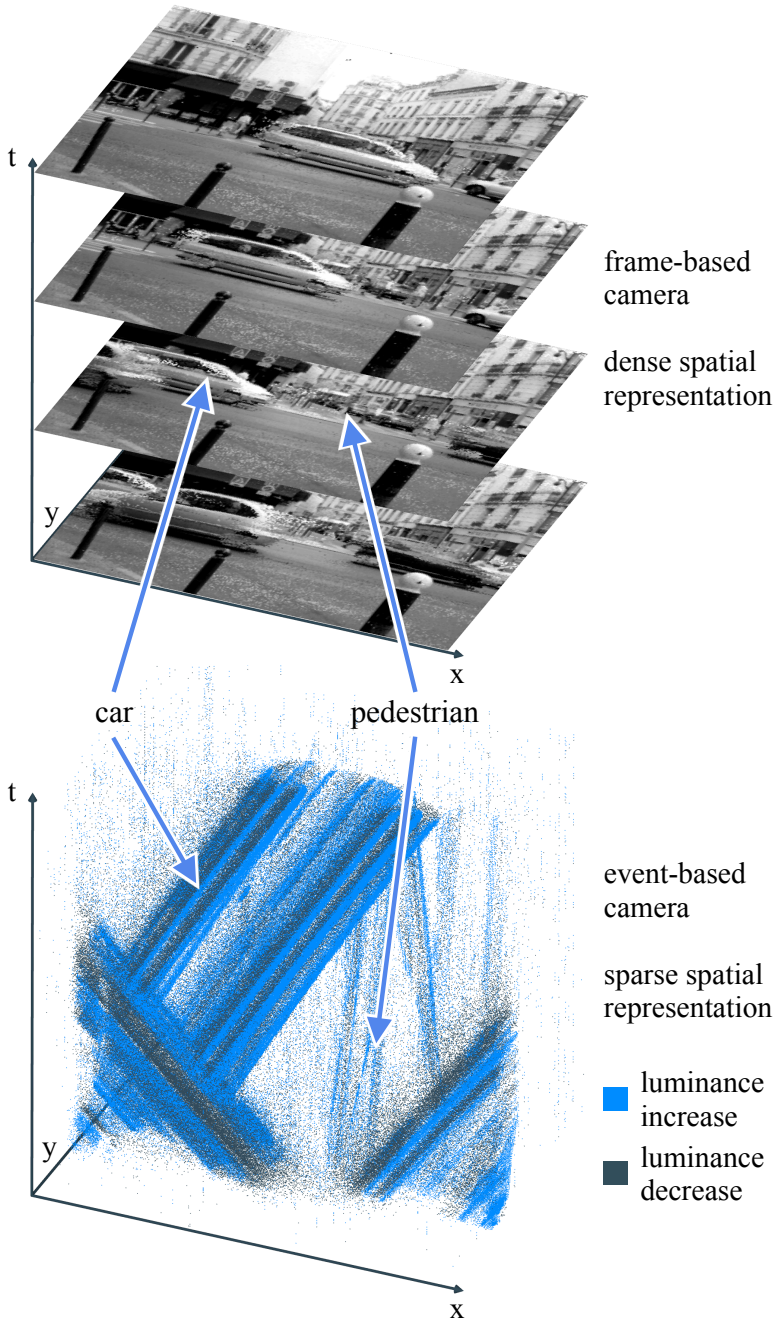


Fig. I.5 *Conventional cameras (top figure) capture dense frames at fixed time intervals. Event-based cameras (bottom figure) have independent pixels which asynchronously output information when the luminance in their individual field of view changes. This sparse representation yields a better temporal resolution and a smaller bandwidth. Some computer vision tasks, such as moving objects segmentation, become easier.*

The next section reviews event-based color pixels and sensors. The ATIS, used in this work to build a color sensor (see chapter III), is given an in-depth description subsection I.2.3.

I.2.2 Event-based color sensors

Several approaches to color sensing have been considered for neuromorphic sensors. In an early attempt, K. Shimonomura et al. combined three silicon retinas to build a three-chip color sensor implementing Land's Retinex theory [82]. Fu et al. [83] introduced a Neuromorphic chip performing color disambiguation: local analog processing is used to determine whether changes detected by the photoreceptors correspond to a variation of chroma or luminance. Similarly, the cDVS pixel [73] detects both brightness changes, as would a DVS pixel, and wavelength changes. However, it does not provide absolute brightness or chroma measurements. This design is also limited by large pixels and poor color separation, as emphasized by Li et al. [84] who introduced the C-DAVIS sensor. The latter combines the DAVIS architecture with a Bayer filter RGB sensor, resulting in an RGBW sensor. The red, green and blue pixels use dedicated photodiodes, whereas the white channel uses the DVS pixel's photodiode. This sensor outputs DVS-like events corresponding to brightness changes and color frames. The Color-DAVIS346 [85] is a DAVIS camera with improved sensitivity, thanks to back side illumination, and a Bayer filter. Thus, it outputs color polarity event, which correspond to changes in color channels, and color frames. A dataset providing recordings of this sensor as well as simulated scenes was recently presented [86].

I.2.3 Asynchronous Time-based Image Sensor

The *Asynchronous Time-based Image Sensor* (or ATIS) [74] is an event-based camera. It features autonomous and asynchronous pixels which generate two types of events, with two photodiodes per pixel.

The first photodiode is associated with a DVS-like circuit, which outputs polarity events when the logarithmic luminance changes beyond a fixed threshold. The polarity bit associated with such events indicates their channel: *on* if the event was triggered by a luminance increase, *off* if it was triggered by a luminance decrease. Upon detecting an event, pixels communicate with the global arbiter which sends events sequentially to the outside world, using an AER interface. Details on the arbiter architecture can be found in K. Boahen's work [71].

The second photodiode and its circuit perform absolute exposure measurements. The measurements are event-based: upon detecting a change, the first circuit (DVS-like) triggers the second circuit (only for its pixel), which starts integrating light received by its photodiode. The time-to-first-spike scheme [87] is used to encode the luminance information. A first event is emitted when the integration begins, and a second event when the circuit's capacitor reaches a specific threshold. With every detection, three events are typically generated by an ATIS: a polarity event (either *on* or *off*), a *first threshold crossing* event shortly after, and a *second* threshold crossing at the end of integration. The absolute luminance is proportional to the inverse of the time difference between the two threshold crossings. Integration restarts if a new change is detected by the first circuit.

The polarity bit of threshold crossing events indicates whether they correspond to the beginning or the end of integration. This approach has two benefits: it yields a very high dynamic range for luminance measurements (143 dB), and it encodes them with spikes, like change detections.

The asynchronous, clock-less ATIS is connected to an FPGA responsible for timestamping events and preparing packets for USB transfer. The two components (sensor and FPGA) form a complete USB camera which exists in two versions: the Opal Kelly ATIS, based on an Opal Kelly XEM6010 board (Spartan-6 FPGA, USB 2.0, 128 MiB RAM used for events buffering), and the CCam ATIS, based on a custom board designed by *Prophesee* (Artix-7 FPGA, USB 3.0, no RAM). The overall ATIS principle and the camera's output are illustrated figure I.6.

Event-based sensors are not the only silicon retina offspring designed to communicate with conventional computers [88]. Other approaches include pulse-modulation imaging [89] and smart vision chips [90, 91, 92, 93]. Such sensors generally output dense frames, and are therefore compatible with conventional computer vision algorithms. Event-based sensors, on the other hand, generate a fundamentally different spare sequence which calls for different computational strategies [68].

I.3 Event-based computer vision

I.3.1 Algorithms

There are three approaches to information extraction from the output of event-based cameras. The first one consists in generating spatially dense frames from the sensor output in a way that preserves temporal resolution. The frames can then be fed to conventional computer vision algorithms [94, 95]. The second approach advocates short calculations triggered by each event, and requires a rethink of computer vision from the ground up [14, 16, 96, 97, 98, 99]. By matching the sensor data format, this approach benefits from the sensor advantages, notably resemblance to biological signals, low latency and data compression. Lakshmi et al. [99] surveyed event-based algorithms for computer vision, and organized them in three categories: object detection and recognition, object tracking, and localization and mapping. Spiking neural networks fit the constraints of the second approach, and several event-based computer vision algorithms were implemented on neural simulators [15, 100, 101, 102]. The third approach mixes frames and events, and is well suited to hybrid sensors such as the DAVIS [103, 104, 105, 106].

Given the issues arising from the Von Neumman architecture of modern computers [107], dedicated hardware seems required for event-based vision systems to match the performance of their biological counterparts. Nevertheless, microprocessors remain the de facto standard to perform general-purpose computations. They benefit from years of research and development, making them cost-effective, computationally-efficient, and user-friendly. As such, they are great tools for algorithms prototyping and early applications of event-based sensors. S. Furber [108] envisions heterogeneity in future processors: general-purpose cores will work together with dedicated hardware accelerators. Under this assumption, a framework targeting CPUs is not a mere temporary solution waiting to be replaced by neural networks, but a decision support tool. It provides a baseline for algorithms power consumption and computational cost, against which implementations

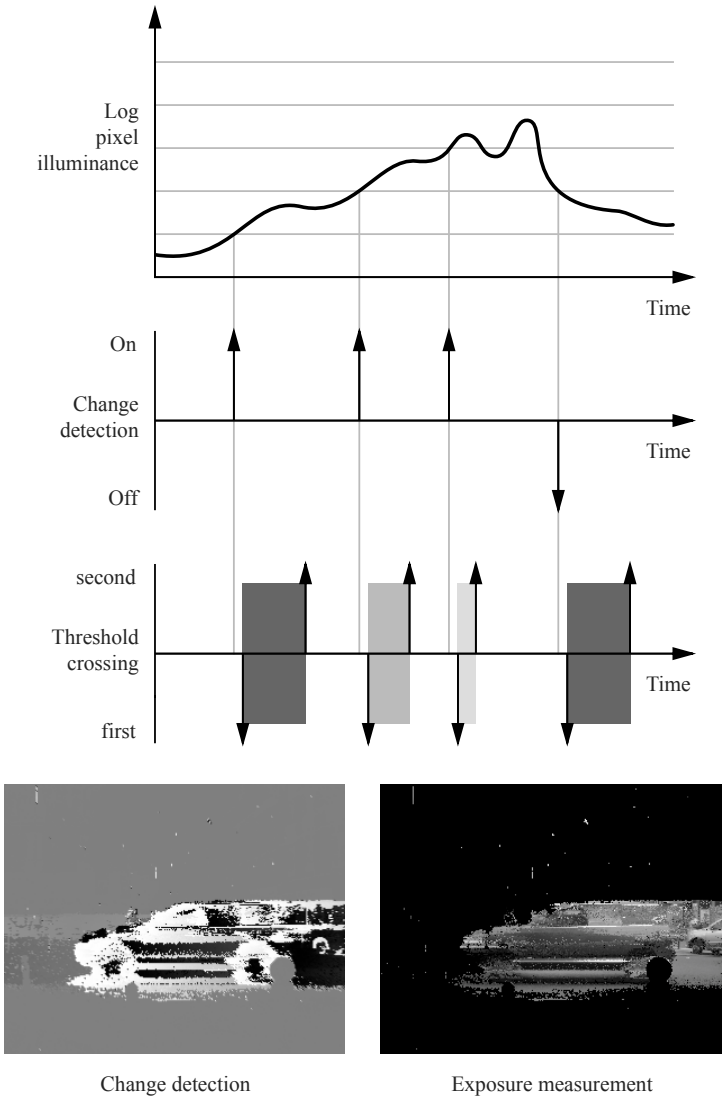


Fig. I.6 *The ATIS is an asynchronous, event-based camera with independent pixels. This figures illustrates the behaviour of a single pixel. When the logarithm of the luminance captured by the pixel crosses a threshold, light integration for the pixel starts. The exposure measurement's duration is proportional to the inverse of the luminance, and is notified by two events called threshold crossings: the first one is sent when the integration begins, the second one when it ends.*

running on dedicated hardware can be compared. Thus, the accelerators can be chosen based on the gain they yield for tasks deemed important. A framework designed for CPUs must provide fast implementations in order to be an effective baseline. Moreover, its syntax should reflect the constrains of hardware dedicated to event-based calculations, to ease comparisons and facilitate algorithms ports from one platform to the other.

I.3.2 Frameworks

A software framework provides a collection of operators and a way to assemble them to build complex algorithms. We consider three types of frameworks related to event-based computer vision. First, we present frameworks for conventional computer vision and their limits when working with event-based data. Then, we examine event-based programming, showing how its concepts apply to event-based computer vision, even though existing frameworks were designed under constraints so different from event-based sensors that they cannot be used directly. Finally, we review frameworks dedicated to event-based computer vision.

The applications of linear algebra to a wide variety of science and engineering fields triggered, early in computer science history, the development of efficient libraries to compute matrix operations [109]. Conventional computer vision libraries use matrices to represent frames, allowing algorithms to be expressed as a sequence of operations on dense data [110, 111, 112]. Dynamic, high-level languages can often be used to specify the operators order. The overhead incurred by the dynamic language is negligible when compared to the matrix operations. The latter are optimized by the underlying linear algebra library, yielding a development tool both efficient and user-friendly. Event-based computer vision is a different story. Small computations are carried out with each incoming event, and the cost of the glue between operators stops being negligible. Hence, the very structure of the libraries designed for conventional computer vision is incompatible with events, besides dealing with dense frames instead of sparse events.

Unlike event-based computer vision, event-driven programming languages and frameworks are not new: Visual Basic dates back to the 1990s. Among the concepts developed for event-driven programming, the event handler pattern and the observer pattern [113] are natural choices to represent event-based algorithms and event-based cameras. Reactive programming [114], devised has a refinement over event-driven programming, introduced new abstractions to avoid state-full event-handlers and explicit time management. However, the neurons we aim at mimicking are state-full (the reaction to an input spike - for example, an output spike - depends on the current membrane potential), and fine control over time management is a needed feature for real-time systems. Hence, we choose to design our framework using event-driven rather than reactive concepts. Modern event-driven frameworks have notable applications in graphical user interfaces and web servers [115], where events represent user interactions and HTTP requests, respectively. The number of events per second reached in these applications is very small when compared to event-based cameras. On the one hand, a user clicking or typing does not generate much more than tens to hundreds of events per second [116], and a large website such as *Twitter* handles about six thousands requests per second on average [117]. On the other hand, an *ATIS* moving in a natural environment generates about one million events per second, with peaks reaching next to ten million events per second. The rel-

atively small number of events the existing frameworks were designed to handle makes their design incompatible with event-based computer vision. For example, Javascript event handlers can be attached or detached at run-time, greatly improving flexibility at the cost of a small computational overhead whenever an event is dispatched.

All the frameworks dedicated to event-based computer vision circumvent the aforementioned problem using event buffers transmitted from operator to operator. The buffers typically contain a few thousand events spread over a few thousand microseconds. A typical operator loops over the buffer and applies some function on each event. The operator output consists in one or several new event buffers, looped over by subsequent operators. The sequence is dynamically defined at run-time, incurring a computational overhead. However, this cost is paid with every buffer instead of every event, becoming negligible as is the case with conventional computer vision frameworks. The first event-based computer vision framework, *jAER* [118], was designed for the DVS and is written in Java. Subsequent cameras and their increased event throughput triggered the development of C and C++ frameworks: *cAER* [119], recently re-factored and renamed *Dynamic Vision* [120] (both from *iniVation*), *kAER*[1] (from *Prophesee*) and *event-driven YARP* [121] (developed for the *iCub*). Table I.1 highlights the design differences between these frameworks. The table also includes *tarsier*, the computation component of the framework presented chapter II. Unlike the other frameworks, it assembles operators at compile-time, suppressing the need for buffers between components, even though event buffers are still used to communicate with cameras or the file system.

Even though a few event-based sensors capture color information, very little research has been carried out on color event-driven algorithm. However, a wide variety of conventional computer vision approaches tackle the problem of color.

I.4 Color computer vision

The first computer vision algorithms for color images were designed as extensions to existing algorithms for brightness - or grey levels - images [122]. They generally involved a coordinate transformation step to separate luminance and chroma, in order to build features invariant to illuminance [123]. Color computer vision co-existed for decades with brightness computer vision, motivated by humans' ability to segment color-less images and the higher computational cost of color algorithms. Moreover, given the digital representation of color as three brightness layers, it is tempting to reduce color to three independent brightness problems. In the early 2010s, fully automated color segmentation remained challenging, with ad hoc techniques required to solve many problems despite the considerable amount of research that had been carried out [18]. The methods for color video segmentation relied on a model describing tracked objects: superpixels [124], graphs [125, 126, 127] or local classifiers [128, 129]. These methods yield robust and accurate results, but require heavy computations. Other methods rely on clustering techniques, especially *Mean Shift* derivatives, to segment colors [130, 131]. The results quality and very high computational costs drove research on speed optimization and complexity

[1] *kAER 0.6*, used in this work, is the lastest version developed by our laboratory and licensed to Prophesee. Newer versions are now developed and maintained by Prophesee and the source codes are for internal use only, hence their performances are not assessed in this work.

name	open source	operators connection	dependencies	communication and execution	event types
tarsier (this work)	yes	compile-time, C++ templates	-	event-wise function calls, single thread	template event types, contiguous memory
cAER	yes	run-time, XML	Boost, libpng, libusb, libuv	event buffers, single thread	hard-coded event types, contiguous memory
kAER	no	run-time, C++ / Python	Boost, OpenCV, Python, Qt	event buffers, constant time intervals, single thread	hard-coded event types, contiguous memory
event-driven YARP	yes	run-time, C++ / XML	libace	IP packets, multiple programs	polymorphic event types, non-contiguous memory
Dynamic Vision System	yes	run-time, XML	Boost, libusb, OpenCV, OpenSSL	event buffers, multiple threads	hard-coded event types, contiguous memory

Table I.1: Various C/C++ frameworks provide tools to build event-based algorithms. Despite an identical goal and programming language, they are build upon very different design decisions. Differences impact users' interaction with the framework and the performance of algorithms implementations.

reduction through structuring of the feature space [132, 133, 134], dynamic bandwidth selection [135] or kernel choice improvements [136]. Despite these optimizations, the large computation costs prevent their use for real-time or low-power applications.

The methods presented so far rely mostly on hand-crafted features. The first use of backpropagation to perform supervised learning in *convolutional neural networks* (or CNNs), started a revolution in computer vision in 2010 [137]. Even though the method was not new [138], the training of networks with many layers had become possible thanks to efficient GPU implementations. This approach outperforms hand-crafted algorithms on many computer vision tasks [139, 140]. CNNs hidden layers represent the original image in spaces with a large number of dimensions. Thus, using color images instead of grey level ones has negligible impact on performance. However, since color improves results in some cases, it is almost always used by CNNs. Every aspect of color processing (coordinate transformation, edge detection...) is expressed as a sequence of convolutions. The associated kernels, which are learned from labelled data, are remarkably similar to the receptive fields of neurons found in the human brain [141]. The trained convolutional kernels of *AlexNet* [142] are illustrated figure I.7.

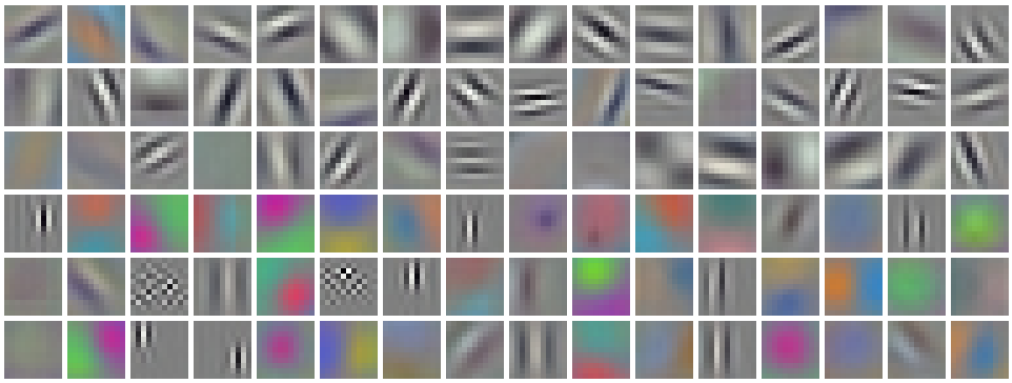


Fig. I.7 *Convolutional neural networks express every computer vision operation as a sequence of convolutions, including color coordinate transformations. The kernel shown in this figure are used by the first layer of the AlexNet algorithm, and were trained on the ImageNet dataset. They are remarkably similar to the recepteive fields of neurons in the human retina and visual cortex. From A. Krizhevsky et al. [142]*

In many ways, today's event-based computer vision is similar to conventional computer vision in the 1980s. Most algorithms are hand-crafted, and very few are designed to manipulate color data [86]. Nevertheless, event-based computer vision has many features that may prove essential to computer vision tasks on embedded system. As a matter of fact, CNNs are much less energy efficient than biological systems, and require large amounts of dense data, which becomes extremely redundant in the case of videos. Neuromorphic sensors, on the other hand, produce sparse events which carry both spatial and temporal information. This property may be critical for efficient video labelling, where both time and space matter. Moreover, they can help solve challenging computer vision problems in complex or unconventional scenes. Such scenes can notably be found when recording biological samples.

I.5 Biology applications

Research in biology is essential to neuromorphic engineering. Even though existing knowledge on the retina and the brain is already a source of inspiration for chip design, there is still much we do not know about the structure of neural circuits and the mechanisms driving them. A deeper understanding may prove necessary to the design of efficient neuromorphic chips. Nevertheless, current chips can participate in biological research, notably to confront theories drawn from observation, often in laboratory settings, to devices built from scratch and used in real-world experiments [143].

Neuromorphic engineering also contributes to biology by taking part in experiments. As a matter of fact, neuromorphic sensors outperform conventional cameras in terms of dynamic range, speed and computational requirements, which benefit biological experiments relying on image acquisition and analysis. They have been considered for calcium-sensitive fluorescence [144], characterized by large difference of luminance between active cells and their background, and quick activity variations. Event-based cameras meet both

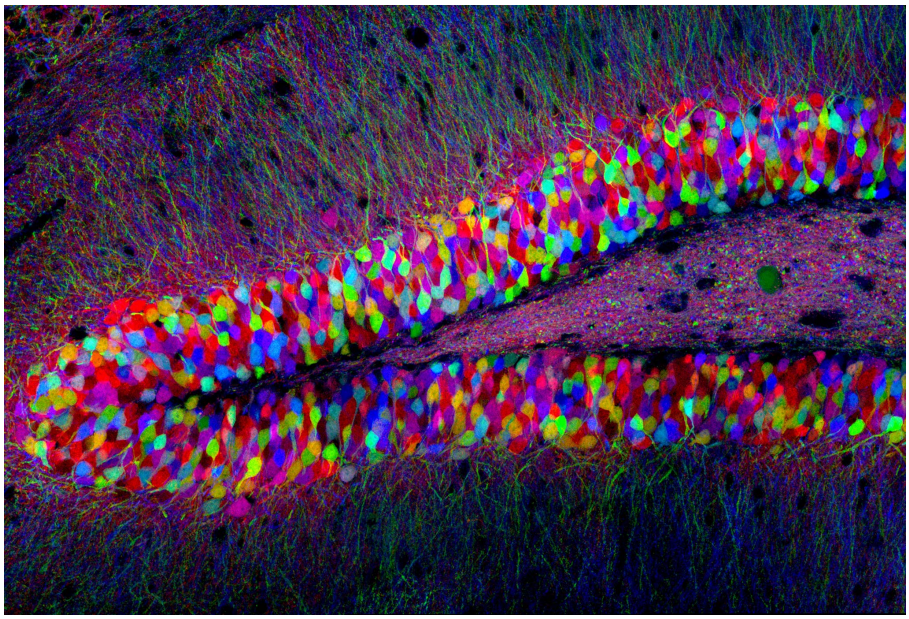


Fig. I.8 *This reconstructed image shows a brain sample of a mouse genetically modified using the Brainbow method. The latter relies on transgenes coding for fluorescent proteins to facilitate neurons segmentation. The gene expression varies widely between neurons, resulting in large luminance variations. The high dynamic range of event-based cameras may be a key advantage to improve Brainbow data acquisition. From T. Weissman et al. [149]*

requirements.

Since neuromorphic sensors aim to mimic their biological counterparts, they are more likely to successfully communicate with the brain. Hence, prosthesis provide another biological application to event-based sensors. The low power consumption of the latter makes them well suited to embedded devices, and the spikes they generate resemble the physical signals expected by biological neurons. Prosthesis using neuromorphic sensors include neural interfaces for spike recording and neuron stimulation [145], sensory substitution using haptic feedback [146], and bionic vision restoration [147, 148].

Brainbow is a bio-engineering method used for neural lineage tracing and neurons segmentation [21]. It relies on multiple transgenes coding for fluorescent proteins. Brain samples created with the method exhibit multiple colors when lit with specific sources, as shown figure I.8. Like calcium-sensitive fluorescence, the Brainbow method may benefit from neuromorphic sensors. We provide an in-depth literature review on this method and considerations on its association with color event-based sensors in chapter V.

Chapter II

Sepia, Tarsier and Chameleon: a modular framework for event-based computer vision

II.1 Introduction

A software framework for event-based computer vision, besides providing guidelines for the design of algorithms, helps to create and distribute efficient implementations. Thus, it is integral to the study of color event-based algorithms and their applications. In order to support color sensors, a framework must be able to represent color events, and must support the synchronization of multiple sensors in the case of a three-chip setup. Existing frameworks for event-based computer vision, such as *cAER* and *kAER*, were designed to handle polarity events generated by a DVS or an ATIS. Both rely on dedicated buffers, resulting in an efficient but not easily extensible implementation. In order to synchronize sensors and manipulate absolute color events - that is, events bundling absolute red, green and blue luminance measurements - one must develop non-trivial extensions to these frameworks. Therefore, it is interesting to design a new framework easing the creation of new event types. Moreover, drastic structural changes can help improve the efficiency, expressiveness and portability of existing solutions. This chapter presents the framework we designed, which features support for color events and the ATIS.

The frameworks components are presented in the order they intervene in a pipeline, starting with an overall view (§ II.2). We introduce event-driven programming concepts and shows how they apply to event-based computer vision (§ II.3), followed by a brief description of *sepia*, the component implementing functions to read and write event files. § II.4 presents the design and implementation of *tarsier*, a collection of event-based algorithms. Benchmarks are used to compare its performance with existing event-based computer vision frameworks (§ II.5). § II.6 describes *chameleon*, a collection of Qt components to display events on a conventional screen. The implementation of drivers to communicate with event-based cameras, non-feed-forward architectures and considerations on parallelism are exposed (§ II.7), before discussing future work and our conclusions (§ II.8).

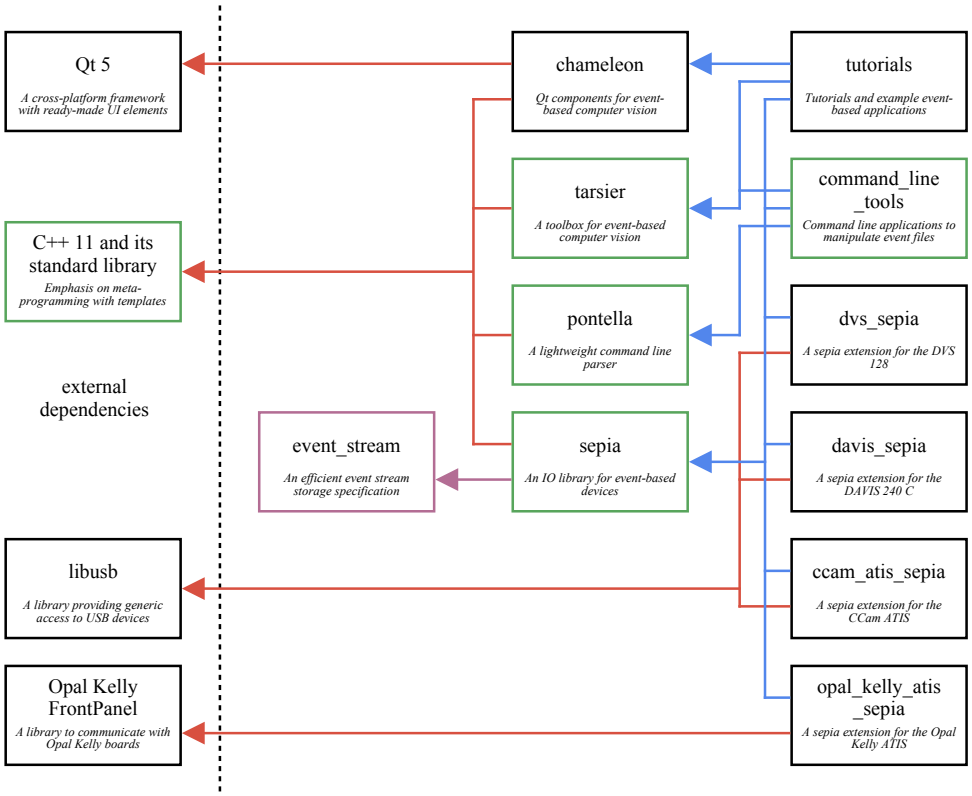


Fig. II.1 *The framework presented in this paper is a collection of three independent components: sepia (file IO), tarsier (event-based algorithms) and chameleon (displays). Each component is hosted on its own repository, and serves a specific goal. This graph shows the three components, their external dependencies, and other repositories dependent on the framework. The event_stream component (purple) is not a library but a file format specification, detailed Appendix A. The components shown in green have no external dependencies but the C++ Standard Template Library.*

II.2 Framework overview

The framework presented in this work supports Linux, macOS and Windows. It is organized in independent components, named after animals with unusual eyes. They work together by following the same conventions, even though they have no explicit link. This structure, illustrated in figure II.1, reduces to a minimum the external dependencies of each component, and promotes modularity. In particular, several components solely require a C++ compiler, facilitating code sharing between various machines and operating systems, and usage with other libraries. The framework's three major components are *sepia* (file I/O), *tarsier* (algorithms) and *chameleon* (display). Since these components are independent, one may use any of them without the others. For example, *sepia* can be used to read and write event files on an operating system lacking Qt support.

The framework's libraries are header-only: they require neither pre-compilation nor system-wide installation, and several versions of the library can co-exist on the same

machine without interfering. Bundling dependencies with algorithms makes projects more likely to keep working over long periods of time without active support, which we believe is a critical factor for research. Moreover, an algorithm and all its dependencies can be shipped in a single zip file, making code easy to share as the supplementary material of a publication (as illustrated by this paper's supplementary material). Header-only libraries also simplify MSVC support for Windows [150], removing the need for GCC ports such as MinGW.

All the code is open-source, and hosted on our GitHub page[1]. Each framework component is hosted on a distinct repository, and documented in the associated Wiki page. More importantly, the *tutorials* repository provides step-by-step tutorials and commented examples to build event-driven applications with the framework.

II.3 Event-driven programming

II.3.1 A generic event-based algorithm

The object-oriented *observer* pattern consist in two constructs: an observable and an event handler. The former dispatches events at arbitrary times, whereas the latter responds to each event with an action. This pattern provides a natural model for an event-based camera (observable) and an algorithm (event handler). It extends to neuron models (for example, integrate-and-fire), though implementing complex networks with feedback and delays - which can change the events order in time - is not straightforward (§ II.7 provides considerations on this topic). Algorithm 1 gives a generic expression of an event-based algorithm under this paradigm.

A framework reflecting this theoretical expression facilitates algorithms implementation. A function (in the programming sense) which takes an event as sole parameter and returns nothing has a syntax close to algorithm 1. Such a function has to mutate a state to do something useful, thus it is not a function in the mathematical sense (it is non-pure).

Algorithm 1: A generic event-based algorithm, or event handler

```
initialize the state
on event do
    instructions mutating the state
end
```

II.3.2 C++ implementation

The typical C++ implementation of the observer pattern relies on dynamic polymorphism: the event handler inherits a generic class, and the observable holds a pointer to an instance of this class. This approach creates overhead for two reasons. On the one hand, every call to an event handler requires a vtable lookup and an extra dereferencing. On the other, the compiler is given less information to optimize the program.

[1] https://github.com/neuromorphic-paris

```cpp
#include "sepia.hpp"
#include <iostream>

void handle_event(sepia::dvs_event dvs_event) {
    std::cout << (dvs_event.is_increase ? "+" : "-");
}

int main() {
    sepia::join_observable<sepia::type::dvs>(
        sepia::filename_to_ifstream("input.es"),
        handle_event);
    return 0;
}
```

Fig. II.2 *This code snippet is the "hello world" program of the sepia library. The function* `handle_event` *prints a plus sign in the terminal on luminance increase events, and a minus sign on luminance decrease events. The main program creates an observable from a file, with the* `handle_event` *function as event handler. This program, provided in supplementary materials, only needs the sepia library in its directory to be compiled on any machine.*

Existing frameworks (cAER, kAER, event-driven YARP and Dynamic Vision System) solve this issue using buffers: events are handled thousands at a time, reducing overhead proportionally. In return, user-written handlers (called *modules* in cAER, Dynamic Vision System and event-driven YARP, and *filters* in kAER) have to loop over buffers of events. Manipulating buffers, though good for performance, may foster practices that deepen the gap with neuromorphic hardware: using events ahead in the buffer to improve performance, as they are "already there", and viewing the events as pieces of data rather than function calls. The former makes the conversion to neuromorphic hardware harder (the algorithm uses future events, increasing memory usage and latency waiting for them), while the latter strips away the event meaning (a model of a hardware spike).

The presented framework relies on static polymorphism, using templates [151]: the event handler is bound to the observable during compilation. This approach does not incur an overhead with every event, therefore buffers are not needed. The algorithm is specified by a loop-free function, illustrated in figure II.2. We want to emphasize that the code presented in this figure is a complete program, which can be compiled without prior libraries installation. The function `handle_event` modifies the state of the `std::cout` object, captured implicitly as a global variable. Events are read from the file `"input.es"`, which uses the *Event Stream* encoding (see the appendix A).

The *sepia* header used in this example implements file IO in the framework, and can be extended to communicate with cameras (§ II.7). Even though it relies on buffers, similarly to the other C++ frameworks, the event loop is hidden from the user. This is meant to reconcile two somewhat paradoxical objectives: provide a fast implementation on CPUs, which work best with bulk data, and encourage an algorithm design likely to translate to highly distributed neuromorphic systems with fine-grained calculations.

Static polymorphism is implemented in sepia using the same approach as the C++ Standard Template Library (see, for example, the *compare* function of the *std::sort* algorithm). Besides being efficient, it allows compile-type, type-safe "duck typing": the code will compile as long as the syntax `handle_event(event)` is valid. Notably, `handle_event`

```cpp
#include "sepia.hpp"
#include <iostream>

int main() {
    auto previous_t = 0ull;
    auto activity = 0.0f;
    sepia::join_observable<sepia::type::dvs>(
        sepia::filename_to_ifstream("input.es"),
        [&](sepia::dvs_event dvs_event) {
            activity *= std::exp((dvs_event.t - previous_t) / -1000.0f);
            activity += 1.0f;
            previous_t = dvs_event.t;
        });
    std::cout << activity << std::endl;
    return 0;
}
```

Fig. II.3 *Unlike figure II.2, this program uses a lambda function to implement an event handler. Lambda functions can be declared inside the main function, keeping the global scope clean. This event handler implements a leaky neuron to compute the activity. The latter is printed once all the event from the source file* `"input.es"` *have been processed.*

can be a function, a lambda function or an object with an overloaded call operator. Lambda functions are great to quickly prototype an event-driven algorithm, as shown figure II.3. This second example is a standalone, dependency-free program as well. The state variables `previous_t` and `activity` are captured by reference in the lambda function. The latter implements a sensor-wide "leaky integrate" neuron to estimate the activity, printed after processing all the input file's events.

The `sepia::join_observable` function blocks until all the events are processed, preventing other routines (notably Graphical User Interfaces) from running. Under the hood, it uses the GUI-compatible `sepia::make_observable` function, which dispatches events on another thread. In turn, this function constructs a `sepia::observable` object. The latter's constructor cannot be called directly, because C++ does not allow class template deduction from a constructor (until C++ 17). Thanks to the *make* function, the event handler type does not have to be explicitly specified. However, the event handler must be statically specified - not unlike connections in a neural network. Changing the event handler at run-time requires an explicit if-else block within the handler.

Both the `sepia::join_observable` and `sepia::make_observable` functions require a template parameter: the expected event type. The event handlers signature is check at compile-time, whereas the file events type is checked at run-time (each *Event Stream* file contains a single type of events).

The event handlers presented thus far have several shortcomings: they use global variables, can be used only with specific event types, and cannot be easily used from other algorithms. The *tarsier* library tackles these issues.

II.4 Building blocks

Basic blocks that can be assembled into complex algorithms are the central feature of a framework for computer vision. They reduce development time and foster code reuse:

components debugged and optimized by an individual benefit the community.

II.4.1 Partial event handlers

In order to represent a building block for event-based algorithms, we introduce the concept of partial event handler, illustrated by the algorithm 2. A partial event handler is triggered by each event, similarly to the complete event handler defined subsection II.3.1. However, instead of consuming the event, the partial event handler performs a calculation, then conditionally triggers a second handler.

Algorithm 2: A partial event handler

```
initialize the state
on event do
    instructions mutating the state
    if condition then
        trigger another event handler with a new event
    end
end
```

Using functions to represent handlers, we denote f_* a partial event handler. Since f_* generates events, it is an observable for a complete event handler g. Binding g to f_* yields the complete event handler f_g. When called, it performs the calculations associated with f_*, then calls g. Any number of partial event handlers can be chained to build an algorithm, as long as the last handler is complete. For example, with g_* now a partial event handler, and h a complete event handler, one can build the pipeline f_{g_h}. For each child, its direct parent is an observable generating events. For each parent, its child is a complete event handler (g_h is a complete event handler and a child for f_*). The syntax can be extended to partial event handlers generating multiple event types: $f_{*,*}$ is a partial event handler with two observable types.

A more common approach to defining algorithms consists in specifying inputs and outputs for each block. However, since a partial event handler conditionally generates (possibly) multiple event types, a generic output is a list of pairs {event, boolean} representing optional objects[2]. Each boolean determines whether the event it is associated with was indeed generated. The program assembling the pipeline would contain a complex sequence of function calls and nested if-else statements to propagate only events that were actually generated. Nested observables yield a syntax both easier to read and more closely related to the event-driven nature of the algorithm.

f_{g_h} is written $f \to g \to h$ in figures to avoid nested indices. Complex pipelines, including merging and feedback, are discussed § II.7.

[2]Using C++ STL primitives, the output's type would be `std::tuple<std::pair<event_type_0, bool>, std::pair<event_type_1, bool>, ...>`. With the C++ 17 standard, `std::optional<event_type>` can be used instead of pairs.

```cpp
#include "mask_isolated.hpp"
#include "mirror_x.hpp"
#include "sepia.hpp"
#include "shift_y.hpp"
#include <iostream>

int main() {
    const auto header = sepia::read_header(sepia::filename_to_ifstream("input.es"));
    sepia::join_observable<sepia::type::dvs>(
        sepia::filename_to_ifstream("input.es"),
        tarsier::make_mask_isolated<sepia::dvs_event>(
            header.width,
            header.height,
            1e3,
            tarsier::make_mirror_x<sepia::dvs_event>(
                header.width,
                tarsier::make_shift_y<sepia::dvs_event>(
                    header.height,
                    10,
                    [](sepia::dvs_event dvs_event) {
                        std::cout << (dvs_event.is_increase ? "+" : "-");
                    })))));
    return 0;
}
```

Fig. II.4 *This program uses both sepia and tarsier. It can be compiled on any computer without installing external libraries. The pipeline is implemented as a sequence of nested partial event handlers.* `tarsier::mask_isolated` *removes noisy events,* `tarsier::mirror_x` *inverts the x coordinate and* `tarsier::shift_y` *shifts the y coordinate by a fixed offset. Events outside the original window after shifting are not propagated.*

II.4.2 *tarsier* implementation

The framework's *tarsier* library is a collection of partial event handlers implemented in C++. Each handler is declared and defined in a single header file: only the included ones are compiled with the program. This organization makes the code resilient to compatibility errors in unused handlers.

The partial handlers are implemented as classes with an overloaded call operator. The children handlers types are templated. In order to allow type deduction, each class is associated with a *make* function: the partial event handler f_* is associated with $make\_f$. For any complete event handler g, $make\_f(g) := f_g$. Pipelines are built by nesting *make* functions: $make\_f(make\_g(h)) = f_{g_h}$. Unlike event handlers, the high-order *make* functions are pure. Most of them take extra parameters to customize partial event handlers. For example, `tarsier::make_mask_isolated`, which builds a partial event handler propagating only events with spatio-temporal neighbors, takes a sensor width and height and a time window as parameters. Figure II.4 shows a simple *tarsier* pipeline, bound to a *sepia* observable.

The *tarsier* and *sepia* libraries are compatible even though they are not explicitly related. Every partial event handler provided by *tarsier* uses template event types, besides template event handlers parameters. The event type has to be specified explicitly (`sepia::dvs_event` in figure II.4), and must have a minimal set of public members which depends on the event handler (often x, y and t). A C++ struct with at least these three fields meets the requirements of most *tarsier* handlers. Users can define custom types

to best represent the events output by their algorithms (flow events, activity events, line events, time surfaces…), or to customize the events payload (with a camera index for stereo-vision, sparse-coding labels…).

This implementation has several benefits. Since the pipeline is assembled statically, type checks are performed by the compiler. Missing event fields and incompatible observable / event handler bindings are detected during compilation, and meaningful errors are returned (in contrast with run-time segfaults). Moreover, an event loaded from disk or sent by a camera, with a specific type, can be used directly without an extra copy to a buffer holding events with another type. Since the compiler manipulates a completely specified pipeline, it can perform more powerful code optimizations. Finally, since static event handler calls have no run-time overhead, events buffers can be traversed depth-first instead of breadth-first (figure II.12). This operation ordering reduces the pipeline latency, as observed in § II.5.

II.5 Comparative benchmarks

Event-based computer vision shows promise for real-time sensing on robots [152]. If a CPU is used to run computer vision algorithms on a robot, the code efficiency can make the difference between a real time and non real time system. Performance is also essential to make realistic comparisons of conventional hardware and neuromorphic hardware, or to compare two event-based CPU algorithms. Even though the average number of operations per event gives an estimation of an algorithm complexity, it does not account for compiler optimizations, memory IO or processor optimizations (branch predicting, cache…). Hence, accurate speed comparisons require a comparison of implementations, whose result depends on the quality of the implementations.

The efficiency of an implementation depends on many parameters, including the algorithm itself, the choice of programming language, the use of suitable programming primitives, and the properties of the framework. We aim to compare the contribution of the latter among frameworks designed for event-based computer vision. We restrict this comparison to frameworks written in C/C++, to avoid comparing languages rather than frameworks. The compared algorithms are given the same implementation in each framework, thus observed differences can only be attributed to frameworks properties.

The present benchmarks focus on event processing: the *tarsier* library is compared to its counterparts in *cAER*, *kAER* and *event-driven YARP*. The other frameworks components (file IO, camera drivers and display) are not considered. Moreover, we were not able to include *Dynamic Vision Systems* in the benchmarks: its current implementation uses multiple threads and circular FIFOs between modules. Modules running faster than their children overflow the FIFO, resulting in silent event loss. Though not critical for real-time applications, this loss biases benchmark results and prevents graceful program termination, which depends on exact event counting. Nevertheless, since the structural design choices of *Dynamic Vision Systems* are similar to those of *cAER*, we expect comparable results. The code used to run the benchmarks is available online[3]. This resource also illustrates the implementation of the same algorithms in various frameworks.

Before each benchmark, we load a specific stream of events in memory. The events

[3] https://github.com/neuromorphic-paris/frameworks_benchmarks

are organized in packets of up to 5000 events and up to 10 ms (a new packet is created as soon as either condition is met), as to mimic the typical output of a camera. We consider two performance indicators. The *duration* experiment measures the total time it takes to read the packets from memory, run an algorithm and write the result back to memory. It indicates how complex a real-time algorithm can be. The *latency* experiment measures the time elapsed between the moment a packet is available and the moment results are written to memory. A packet is made available when the wall clock time goes past the timestamp of its last event. A busy-wait loop is used to wait for the wall clock time if the framework is ready to handle a packet before the latter is available. This mechanism simulates the output of an actual event-based camera while avoiding putting processes to sleep, which is a source of non-deterministic variations in the measured latency. The packets contain `sepia::dvs_event` objects, chosen as a neutral type for all the frameworks. Event type conversions, if needed, are taken into account in the performance measurement. This choice is not an unfair advantage to *tarsier*, since its handlers are compatible with any event type (including the types provided by *sepia*). The events dispatched from one partial event handler to the next are framework-dependent. However, to avoid uneven memory writes, the output events are converted to a common type before being pushed to a pre-allocated vector. To make sure that the output is not skipped by the compiler as an optimization, we calculate the MurmurHash3 [153] of each output field once the algorithm completed. The resulting values are controlled for each benchmark run, and guarantee that each implementation calculates the same thing.

The benchmarks use five distinct algorithms (p_1 to p_5) described figures II.5 and II.6. Each pipeline is assembled from one or several of the following partial event handlers:

- `select_rectangle` only propagates events within a centered 100×100 pixels window.
- `split` only propagates events representing a luminance increase.
- `mask_isolated` only propagates event with spatio-temporal neighbors.
- `compute_flow` calculates the optical flow.
- `compute_activity` calculate the pixel-wise activity. The activity decays exponentially over time, and increases with each event.

We use three event streams, listed in table II.1 and available in the benchmarks' repository. These streams contain polarity events recorded by an ATIS, in both controlled and natural environments. The *duration* experiment is run one hundred times for each combination {stream, pipeline, framework}, and the *delay* experiment ten times. Each *delay* task generates many samples, whereas each *duration* task yields a single value. All 6600 tasks are shuffled, to avoid possible biases, and run sequentially on a computer running Ubuntu 16.04 LTS with an Intel Core i7-6700 CPU @ 3.40GHz CPU and a 16 GB Hynix/Hyundai DDR4 RAM @ 2.4 GHz. The code is compiled with GCC 5.5, C++11 and the -O3 optimization level.

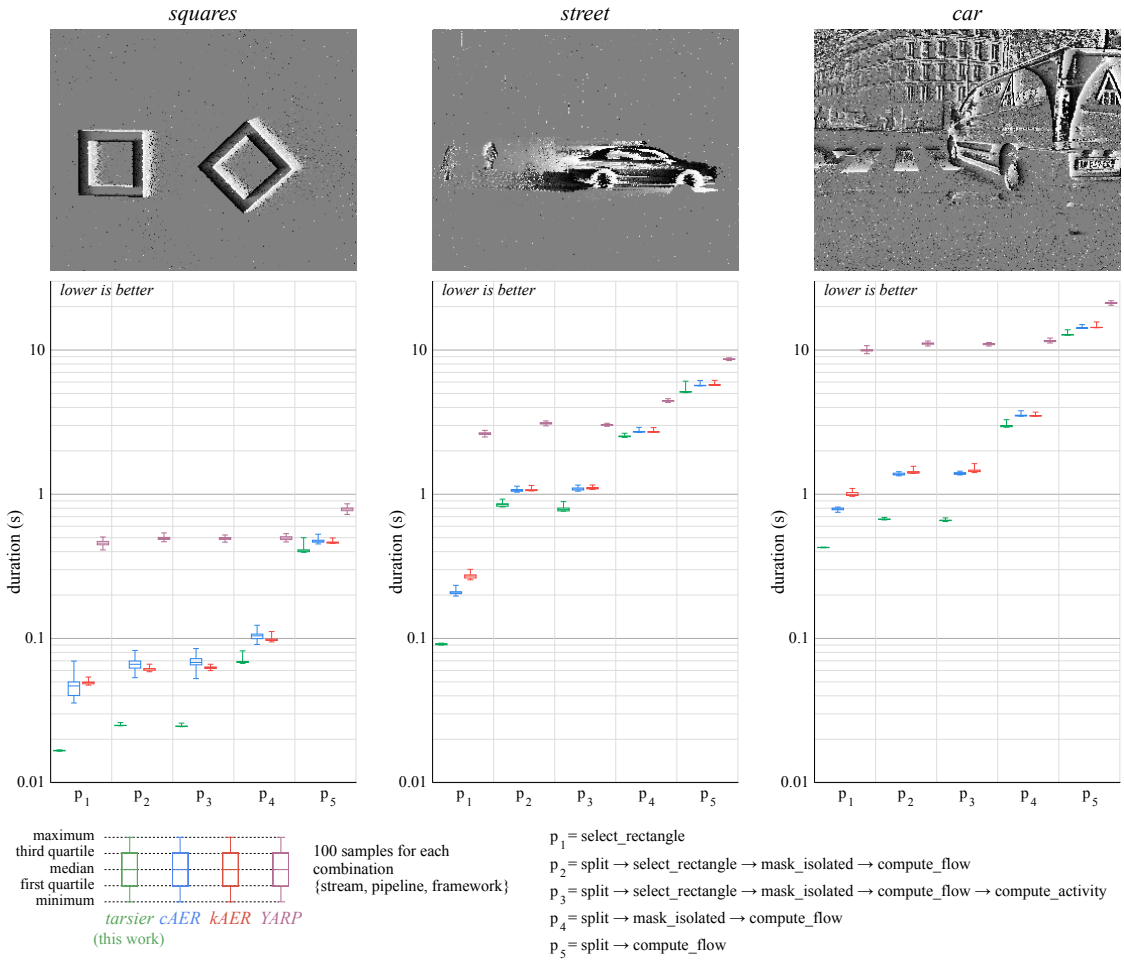


Fig. II.5 *We implement the same partial event handlers in each framework in order to compare them. We consider five pipelines and three event streams. The total time it takes to handle every event from the input stream is measured one hundred times for each condition. We attribute the better performance of tarsier to static polymorphism, which yields a program with fewer memory operations. The poor performance of event-driven YARP can be attributed to non-contiguous memory storage.*

stream name	description	duration (s)	event rate (s^{-1})
squares	artificial scene, moving geometric shapes, fixed sensor	9.50	2.83e5
street	natural scene, moving pedestrians and cars, fixed sensor	50.6	3.17e5
car	natural scene, sensor inside a moving car	69.6	9.56e5

Table II.1: We use three event streams recorded by an ATIS to perform benchmarks. The streams were chosen for their different conditions (artificial and natural scenes, fixed and moving sensor) and average event rates.

II.5.1 Duration

The *duration* benchmark results are illustrated in figure II.5. The approach presented in this paper yields the smallest duration on all the pipelines and event streams considered. This improvement over state-of-the-art frameworks can notably be attributed to a reduced number of memory reads and writes, thanks to the template event types.

The performance of *event-driven YARP* is substantially worst than the performance of the other frameworks. This gap is most likely related to the non-contiguous buffers used by this framework. Others use either hard-coded event types (*cAER*, *kAER*) or template event types (*tarsier*) to leverage contiguous memory.

The pipeline p_3 contains more operations than p_2. Yet, the p_3 *tarsier* implementations has a smaller duration than p_2 (the effect is most visible with the *street* stream). The `compute_activity` event handler does not utilize the visual speed calculated by `compute_flow`, only the flow events' timestamp and position. Therefore, the flow computation can be skipped without changing the algorithm outcome. In the case of frameworks with modules assembled at run-time, the compiler cannot make this simplifying assumption. We believe this behavior can improve the performance of complex pipelines, where finding redundant or unused calculations manually can prove difficult.

II.5.2 Latency

The *latency* benchmark results are illustrated in figure II.6. Wall clock time is measured with microsecond precision for each input packet and each output event. Latency samples are calculated by subtracting the wall clock time of output events and that of their input packet. In some cases, the latency is zero, meaning that the actual elapsed wall clock time is smaller than the measurements' precision. To allow representation on a log-scale, we round up null latency samples to 0.5 µs.

The relative standard deviation is much higher for the *latency* benchmark than the *duration* one. As a matter of fact, measured values are much smaller: durations are

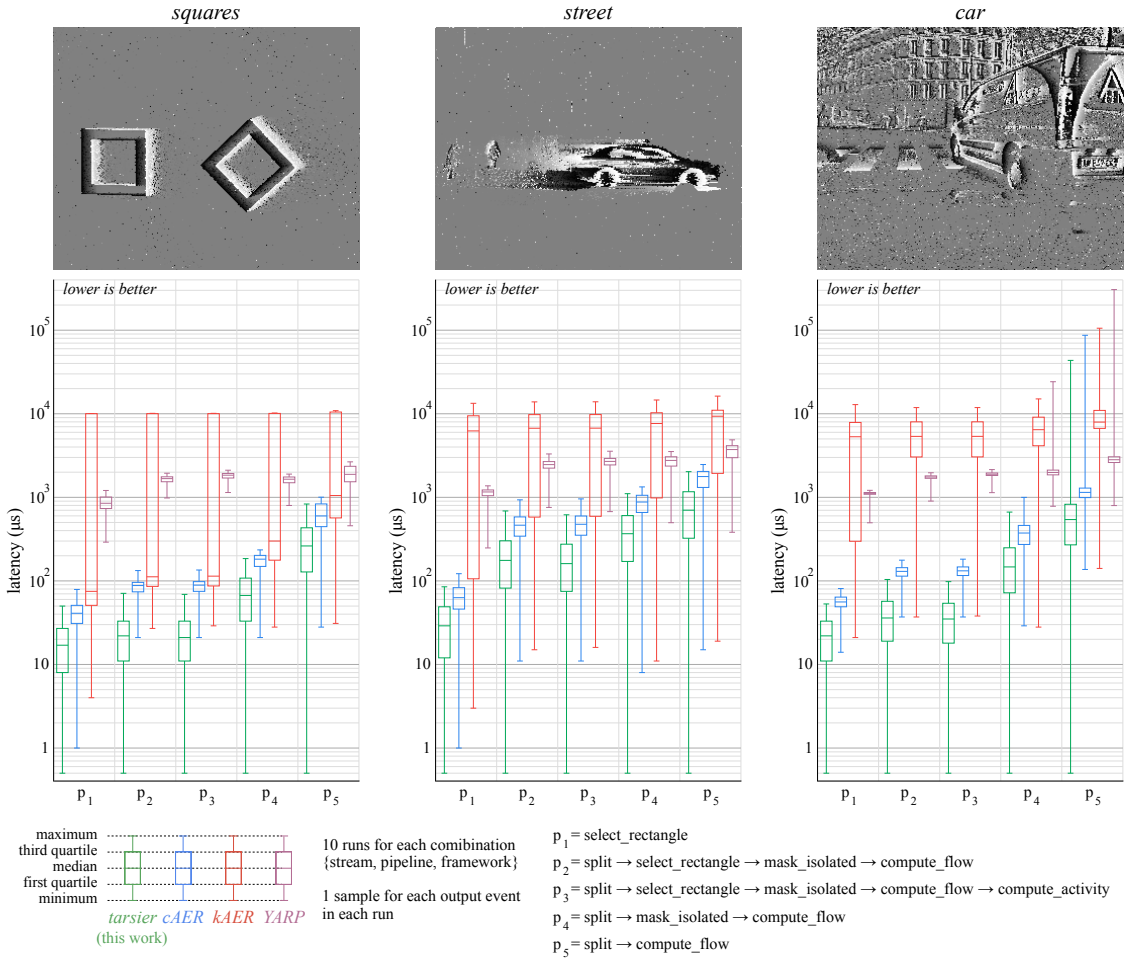


Fig. II.6 *Low-latency is an important feature of event-based cameras, and therefore event-based frameworks. We measure the time elapsed between the moment a buffer is available and the moment associated output events are produced by the pipeline. Events that are not propagated by the pipeline (for example, removed noise) are not taken into account. For each condition, latency is measured for each output event over ten runs of the whole stream. We attribute the better performance of tarsier to depth-first traversal. kAER under-performs in this benchmark since it constrains buffers duration, unlike the camera model assumed in the benchmarks: the resulting buffer reorganization increases delays. This benchmark's relative variations are larger than the duration benchmark's variations. The same time measurement functions are used, however durations are order of magnitude larger than latencies.*

in the order of seconds, whereas latencies are on the order of microseconds. Thus, every small non-time-deterministic operation (memory operations, CPU prediction, kernel preemption...) has, relatively, more impact.

The *kAER* framework yields substantially larger latencies than the other frameworks. Since it enforces buffers with a constant duration, latency increases when the buffers provided by the camera use a different, possibly variable, duration.

The framework presented in this paper outperforms the others in this benchmark as well. Low-latency can be a major benefit for robots or closed-loop systems. The performance gain is a consequence of buffer depth-first traversal and the reduced number of memory operations, since inter-handler communication is not implemented with buffers. The latency reduction improves with the duration of the algorithm when comparing *tarsier* and *cAER*, as illustrated in figure II.7 (top graph).

However, the latency variance is larger for *tarsier* than *cAER*, and increases with the pipeline duration as well. This is another consequence of depth-first traversal: the first event in the input buffer is handled as soon as the packet is available, and therefore has a small latency. In contrast, the last event in the buffer waits for all the other events to be handled, resulting in a much larger latency. This phenomenon does not exist with *cAER* since the whole packet is processed by each module sequentially: events with the same input packet exit the pipeline at the same time.

The latency used so far takes only the framework into account. The first event of each buffer is also the one that waited the most in the camera while the input buffer was being filled. If we neglect the USB transfer duration, we can define the total latency associated with an event as the sum of the framework latency and the timestamp difference between the last event in the packet and the considered event. The total latency as well as its variance are both smaller for *tarsier* when compared with *cAER*, since the packetization effect is counterbalanced by the depth-first traversal. Both the framework latency and total latency densities are illustrated in figure II.7 (bottom graphs).

II.6 Event displays

Conventional screens display frames at fixed time intervals[4]. In order to display events, one has to perform a conversion. Most frameworks rely on fixed time windows: a frame pixel is colored in white if it was the source of a luminance increase event during the associated time interval, in black if the luminance decreased, and in grey if nothing happened. This approach does not account for the high temporal resolution of the signal. Another method relies on time decays [154, 155]: the frame pixel i is given the color $c_i = \frac{1}{2}\left(1 + \delta_i \cdot exp\left(-\frac{t-t_i}{\tau}\right)\right)$. t is the current timestamp. t_i is the timestamp of the most recent event generated by the pixel i. $\delta_i = 1$ if the last event generated by i corresponds to a luminance increase, and -1 otherwise. τ is a fixed decay. Figure II.8 illustrates the difference between the two methods, highlighting the benefits of exponential decays.

The full-frame decay rule requires an exponential calculation on every event for every pixel (for an ATIS, 72960 pixels a million times per second), which is both unrealistic

[4]Recent screens compatible with Nvidia's G-Sync technology can display frames at varying time intervals, narrowing the gap between frames and events. Exponential decays can be used to convert events to frames compatible with such screens.

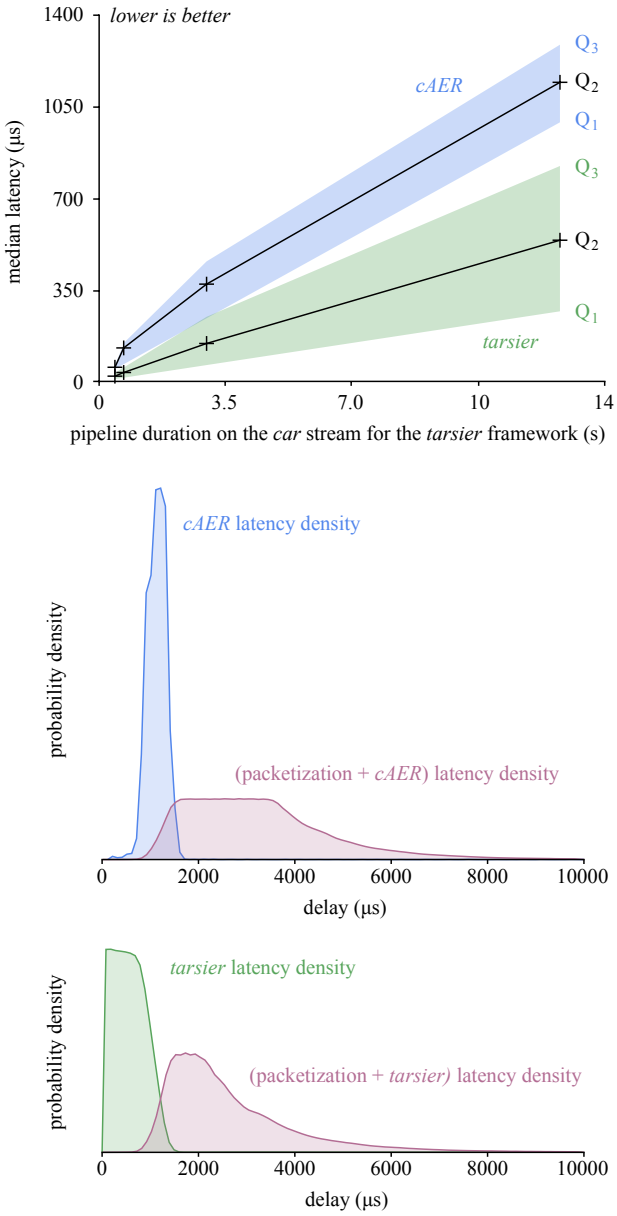


Fig. II.7 *The graphs presented in this figure take a closer look at the latency created by tarsier and cAER for the car stream. In the top graph, latency is plotted as a function of pipeline duration when run with tarsier (arbitrarily chosen as a complexity indicator). tariser has a smaller median density, but a larger variance. The density probability for the most complex pipeline is plotted in the middle and bottom graphs (blue and green). It accounts only for framework latency (as does the first graph). Adding the latency caused by packetization in the camera (before the USB transfer) yields the total latency. The depth-first traversal leveraged by tarsier better counterbalances packetization, resulting in both a lower total latency and a smaller variance.*

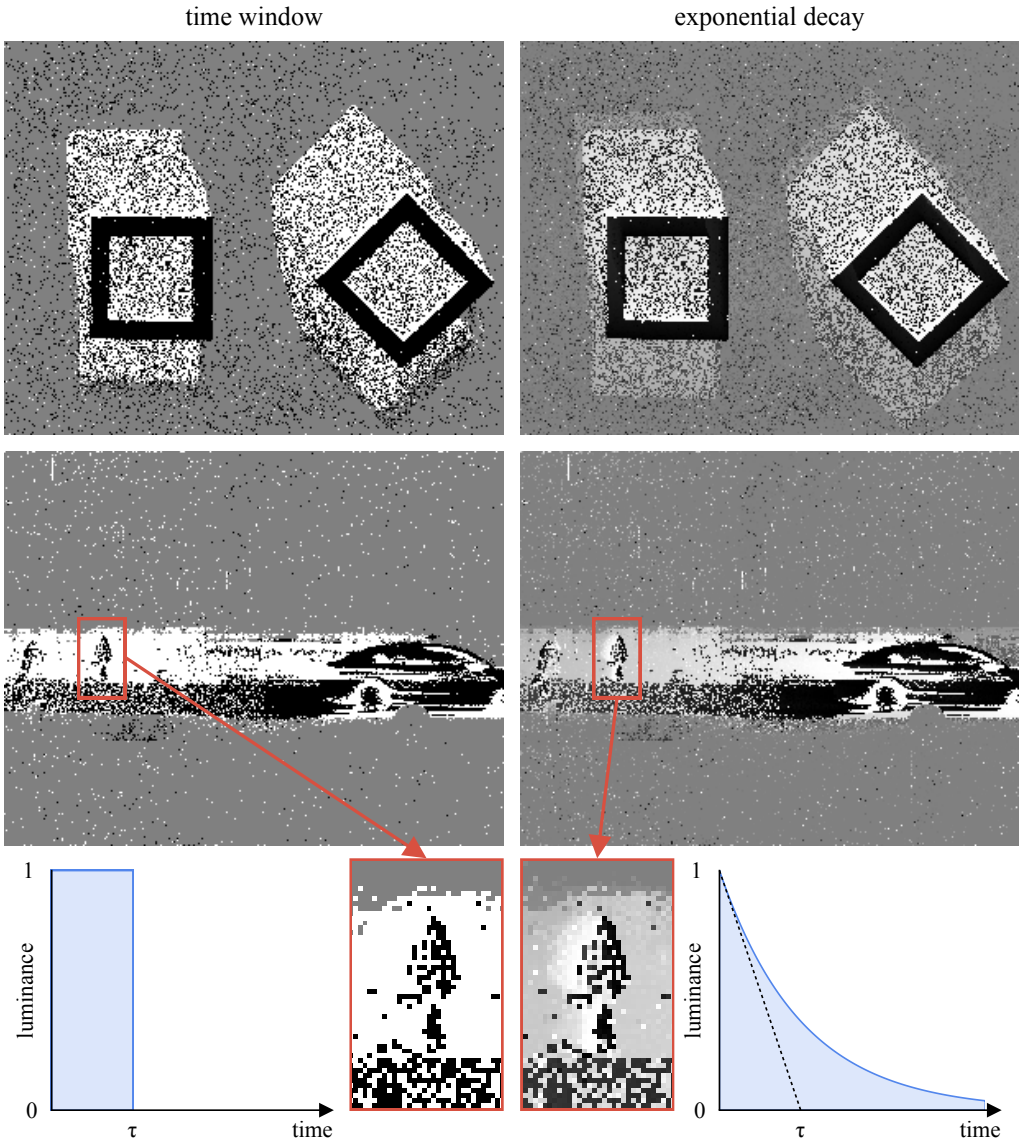


Fig. II.8 *This figures compares two strategies to convert events to frames for display. The time window approach (left) degrades temporal information: the still frames do not hold enough information to determine the geometric shapes motions (top row) or the relative speed of the car and the pedestrian (bottom row). The exponential decay approach (right) represents temporal information with grey levels. It is computationally more expensive than the time window approach, but can be easily implemented on a GPU to relieve the CPU.*

and unnecessary, since the typical display features a 50 Hz refresh rate. Instead, one can calculate the decays only when a frame is about to be rendered, and use the GPU available on most machines to do so. GPUs are designed to run massively parallel calculations with every frame, thus are well suited to this task.

The *chameleon* library provides [156] components to build event displays. The components are independent and header-only. Unlike *sepia* and *tarsier*, *chameleon* cannot be used without *Qt 5*. In return, the event displays can easily be integrated into complex user interfaces. The `chameleon::dvs_display` implements the full-frame decay method mentioned previously. This component assumes two threads: an event loop (for example, a *sepia* observable followed by a *tarsier* pipeline) and a render loop. The loops communicate using a shared memory with one cell per pixel, where the last timestamp and polarity of each event is stored. When a new frame is about to be rendered, the shared memory is sent to an OpenGL program to compute each pixel's time decay. The shared memory is accessed millions of times per second by the event loop. Usual mutexes can cause non-negligible overhead, since they rely on system calls. The *chameleon* implementation uses spin-lock mutexes instead (essentially busy-wait loops with atomic variables), at the cost of increased CPU usage. To minimize the strain on the event loop, the render loop first creates a local copy of the shared memory, then releases the mutex, and finally communicates with the GPU. This mechanism is illustrated in figure II.9. Figure II.10 gives an overview of an application build with the three major components of the framework, with a focus on thread management. This application's code is available in the *tutorials* repository.

The proposed approach does not rely on pre-defined frame boundaries: the frame-rate matches the display rate regardless the event loop speed. Consequently, the visual animation remains smooth even if the event pipeline is slower than real time. A smooth slow-motion display can be created by artificially slowing down the event loop.

The colors used by the DVS display can be customized: the c_i value is then used as a weight parameter for mixing the colors. Transparent colors can be used, enabling display overlays for cameras generating multiple stream types (such as the ATIS or the DAVIS). Other notable components provided by *chameleon* include a vector field display (well-suited to flow events), a blob display, a time delta display (to represent the absolute exposure measurements of an ATIS), and a screen-shot component to easily create frame-based videos. These components use template event types, similarly to *tarsier* event handlers, and the type requirements follow the same conventions. The displays coordinates system follows the usual mathematical convention, with the origin located at the screen's lower-left pixel. The usual computer vision convention (origin at the upper-left pixel) is not used as it is a result of the matrix representation of frames, which event-based algorithms aim to avoid.

II.7 Framework extensions

II.7.1 Camera drivers

Since most event-based cameras feature a USB interface, their drivers can be devised as user-space programs atop a third-party library overseeing the USB communication. To keep the codebase modular and minimize dependencies, each camera interface is held in

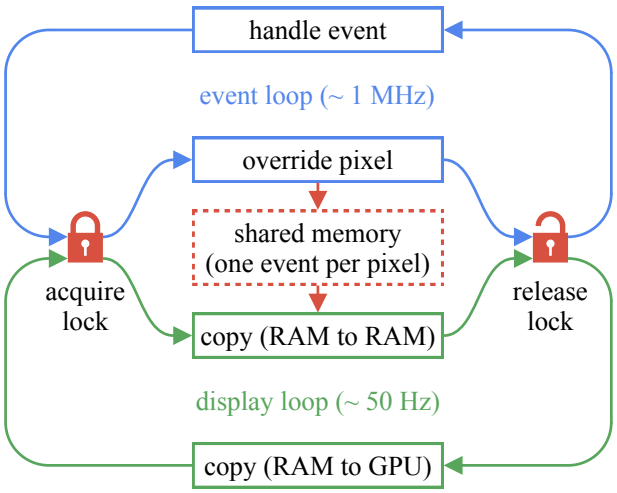


Fig. II.9 *In order to convert events to frames, one has to reconcile the very different rates of the event loop (about* 1 MHz*) and the display loop (often* 50 Hz*). We use a shared memory the size of the sensor, protected by thread-safe locks. On each event, the first thread (blue) overwrites former events with the same spatial coordinates. Every time a frame is about to be rendered, the display loop (green) copies the shared memory to RAM and releases the lock, then communicates with the GPU. The memory-to-memory copy minimizes lock ownership, to avoid blocking the event loop. The lock, acquired with every event, is implemented as a spin-lock mutex.*

a distinct repository extending the *sepia* library.

As of now, the following cameras are supported:

- DVS 128. We re-implemented the *libcaer* interface to provide out-of-the-box MSVC support.
- ATIS (Opal Kelly board). This extension depends on the non-free Opal Kelly Front Panel library.
- ATIS (CCam 3). This camera has the same pixels and arbiter as the Opal Kelly ATIS, however it features a custom FPGA and a USB 3 interface. It was designed by Prophesee.
- DAVIS 240C. We re-implemented the *libcaer* interface for this sensor as well.

Event-based cameras have internal buffers to store events while waiting for a USB transfer. A camera generating events at a faster rate than what the computer program can handle ends up filling its internal buffers to capacity. At this points, camera either drop new events or shuts down. To circumvent this issue, each *sepia* extension uses an extra thread to communicate with the camera, independently of the event loop executing the algorithm. The two threads communicate with a thread-safe circular FIFO. An overall view of the threads of an application using a *sepia* extension, *tarsier* and *chameleon* is given figure II.11. The circular FIFO implementation is provided by *sepia*.

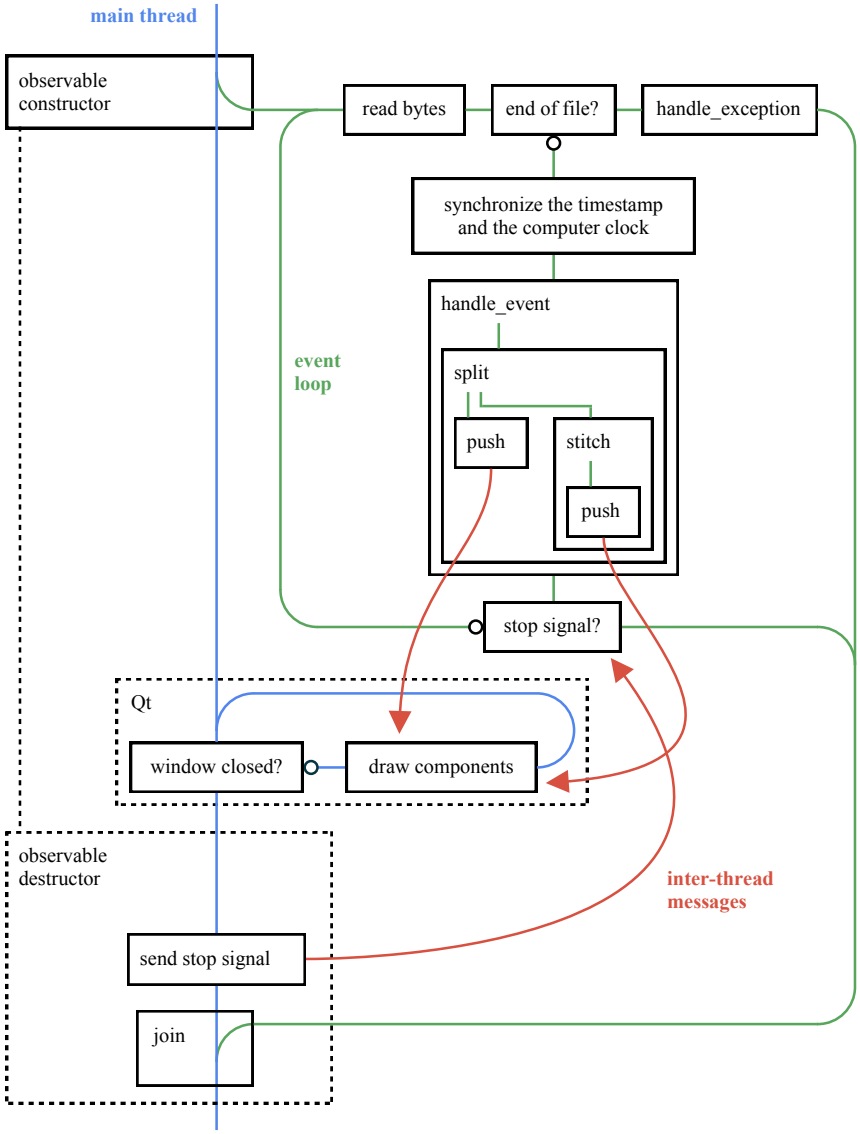


Fig. II.10 *This figure provides an overall view of the threads in an ATIS event stream viewer application using sepia, tarsier and chameleon. The application reads an Event Stream file and displays it as frames. The observable constructor (respectively destructor) creates (respectively joins) the event loop thread, in accordance with the RAII (Resource acquisition is initialization) philosophy of C++. The push inter-thread messages rely on the mechanism illustrated in figure II.9, whereas the stop signal is implemented as an atomic boolean. The code for this application can be found in the tutorials repository.*

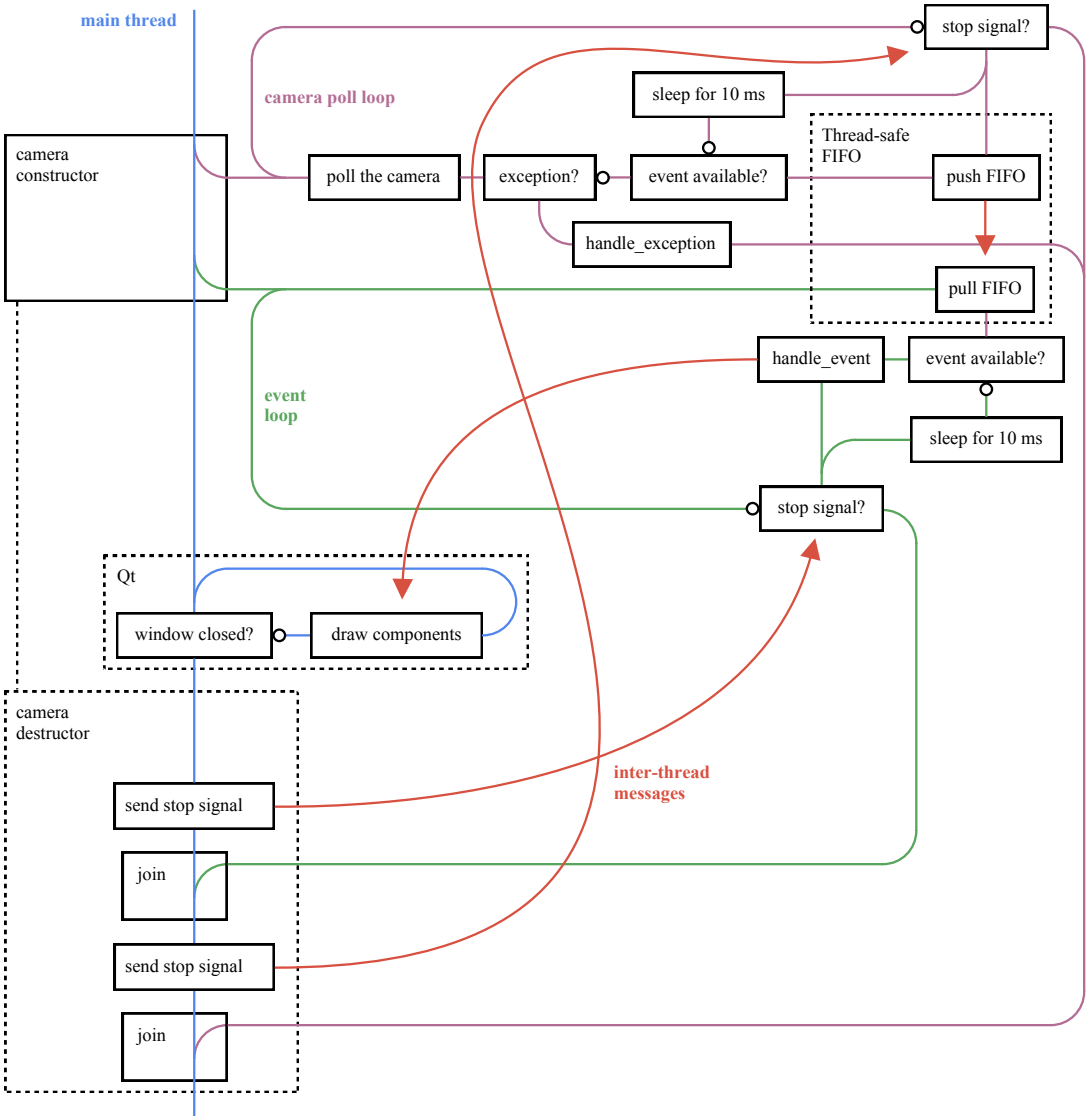


Fig. II.11 *This figure provides an overall view of the threads in an ATIS camera viewer application using opal_kelly_atis_sepia, tarsier and chameleon. The application listens to a camera and displays the generated events as frames. It encompasses the application illustrated in figure II.10. The extra thread is used to communicate with the camera as fast as possible even when the event loop is busy. The two threads communicate through a thread-safe FIFO buffer implemented in sepia. The Opal Kelly Front Panel library does not provide a poll function, hence the explicit sleep step in the graph. However, this function is used by sepia extensions based on libusb, resulting in reduced CPU usage.*

Multiple parameters can be specified to configure an event-based camera, such as the operating mode or the current biases. JSON files are used by *sepia* extensions to specify the configuration. The *sepia* header implements a JSON parser and validator to load configuration files and warn users in case of syntax errors or unknown parameters.

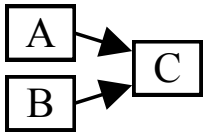
II.7.2 Complex pipelines

The present framework is designed to implement feed-forward pipelines, with optional splits. Most partial event handlers can be represented with populations of neurons, as they perform small calculations with each input. Thus, event-based pipelines can be translated to neuromorphic hardware, though a method to actually perform the conversion has yet to be devised.

However, not all neural networks can be represented with event handlers. Notably, neurons with second order dynamics and synapses with delays dispatch events that are not an immediate response to an input spike. The present framework - and, more generally, purely event-based algorithms - cannot implement such models. To use complex neurons to process the output of a camera, one needs to leverage frameworks designed to implement neural networks. The present framework can, in this case, be used to communicate with sensors, perform low-level processing and send events to the neural network.

Nevertheless, two types of architectures more complex than feed-forward pipelines can be implemented in our framework: streams merging and feedback loops. Even though they still impose more constraints than generic spiking neural networks, they allow for the efficient implementation of algorithms on a CPU without the need for another framework.

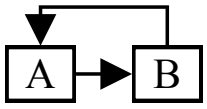
Streams merging has the following generic structure:



A, **B** are partial event handlers, and **C** is a complete event handler. This structure appears when merging the results of several calculations with a common origin. For example, one may split a stream of polarity events to compute two optical flows (one per polarity) and merge them to calculate an overall flow. **A** and **B** run sequentially in this scenario, therefore events are dispatched to **C** in the order of their timestamps. This scenario can be implemented by constructing **C** before the pipeline. The partial event handlers **A** and **B** are both given a reference to **C** as complete event handler. The `std::reference_wrapper` class can be used to prevent template deduction to a non-reference type, which would trigger a copy.

The merge operation can also arise from the use of multiple sensors, for example for stereo-vision or audio-video fusion. In this case **A** and **B** run in parallel, on different threads. Given the non-deterministic scheduling of most operating systems, **C** must re-order the events dispatched by its observables before handling them. This operation is implemented by the partial event handler `tarsier::merge`, compatible with an arbitrary number of observables.

A simple feedback loop can be modelled as:



A and **B** are both partial event handlers. This structure can be useful for flow control or learning. The feedback operation can be executed at various moments in the lifecycle of an algorithm: after processing a batch of data, immediately after each event and after each event with a delay. The implementation of the second and third approaches is not straightforward with existing packet-based frameworks. The whole packet has already been processed by **A** when the first event is processed by **B**, preventing the associated feedback from affecting the next events. The second approach can be implemented in *tarsier* using variables shared between **A** and **B**. Before handling an event, **A** reads from the variables and processes the event accordingly. After handling an event, **B** writes to the variables. Since an event is completely handled before the next is considered, modifications of the shared variables caused by the event n will be available to event $n + 1$. The third approach - adding delays to the feedback - can be implemented by combining the second approach and a merge structure.

II.7.3 Parallelism

The application illustrated in figure II.11 relies on multiple threads, and can take advantage of CPUs with a few cores. However, the sequential strategy presented so far does not harness the full potential of many-cores architectures.

The creation of parallel tasks and inter-task communication have a cost. An application using multiple tasks must reach a compromise on grain size [157]. A large grain size yields less overhead, whereas a small grain size fully utilizes the CPU capabilities. The atomic tasks of an event-based pipeline are its partial event handlers. Larger grain sizes can be obtained by combining several partial handlers into a single task. The tasks represented figure II.12 can be combined either vertically (one thread per event) or horizontally. The former requires inter-thread communication with every partial handler to ensure sequentiality, cancelling the benefits of parallelism, whereas the latter corresponds to the buffer-based approach of *event-driven YARP* and *Dynamic Vision System*. Consequently, latency increases with the grain size.

Parallelism can be beneficial when high latency is not critical and a high throughput is required. However, implementing parallelism efficiently is not straightforward: to avoid FIFO overflows between modules, possibly complex flow control algorithms must be implemented. High-quality libraries provide high-level tools to build parallel algorithms, such as Intel Threading Building Block's flow graph [158]. The partial event handlers provided by *tarsier* can be integrated with such tools. Thus, one can implement an algorithm once and use it with either a low-latency *tarsier* pipeline or a high-throughput flow graph. An example integration of a partial event handler in a class manipulating buffers is given in the *tutorials* repository. This approach can also help integrating *tarsier* with other event-based frameworks, in order to use existing drivers and viewers.

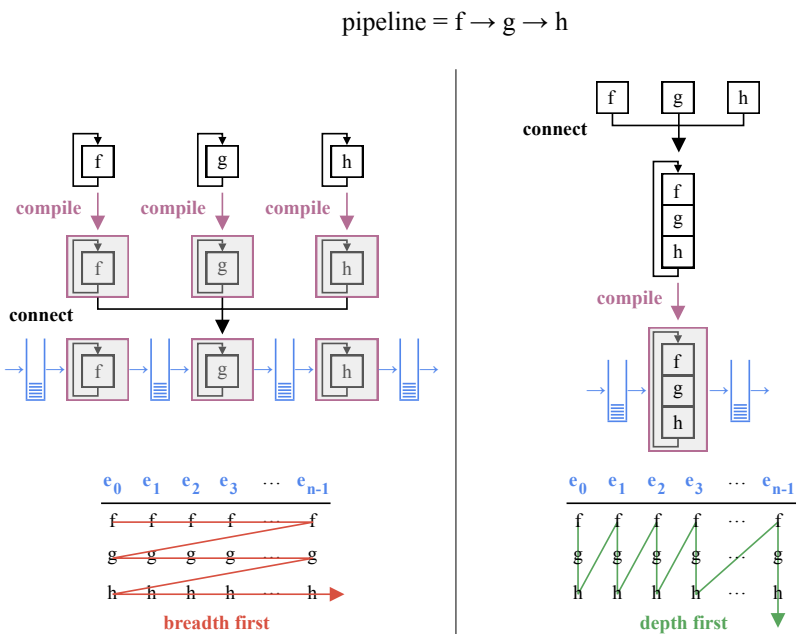


Fig. II.12 *The $3 \times n$ operations associated with a sequence of three event handlers f, g and h and a buffer of n events $e_i, i \in [0..n-1]$ can be performed in two orders: breadth first and depth first. An implementation relying on dynamic polymorphic incurs an overhead for every distinct function call, and must therefore use the breadth first approach (left). Depth first yields lower latencies, but requires static polymorphism: the pipeline must be assembled during compilation (right).*

II.8 Conclusion and discussion

We have presented a modular framework for event-based computer vision with three major components: *sepia*, *tarsier* and *chameleon*. The components, though designed to work together, have no explicit relationship, thus minimizing the external dependencies of each component. Moreover, each component can easily be replaced with other libraries.

The presented framework hides buffers from the user, serving our goal: encouraging functional, event-based semantics likely to translate to neuromorphic hardware while providing an efficient implementation on CPUs. Benchmarks show an increased throughput and a reduced latency compared to state-of-the-art frameworks for event-based computer vision. Using contiguous memory to store events is crucial to performance. Moreover, assembling pipelines before compilation reduces latency and improves throughput, thanks to better compiler optimizations and fewer memory operations. The common practice of hard-coding simple operations (mirroring the stream, removing noise...) in file readers to reduce latency is no longer required with static polymorphism, yielding a cleaner, more generic codebase.

The benchmarks compare performance with pipelines of varying complexity. However, all the considered experiments use simple pipelines (without merges or loops), focus solely on the algorithm performance (the performance of IO and display operations is not

evaluated), and run in real-time on the test machine. In a future work, we plan to devise new benchmarks to cover more use-cases. Moreover, adding more measurements, such as power consumption, will enable comparisons with neuromorphic hardware.

Assembling a pipeline before compiling requires meta-programming - that is, another programming language to generate the actual code. The framework presented in this work uses C++ template meta-programming, since this language is supported by every standard-compliant compiler. Nevertheless, it can be unsettling to new users, and makes the creation of wrappers in high-level languages, such as Python, difficult. A high-level language or graphical user interface must bundle a C++ compiler to generate *tarsier* pipelines. Nevertheless, the framework modular structure and its independence from third-party libraries make it a good candidate for a common low-level library to multiple high-level interfaces. It can notably be integrated with native Android applications, or used to speed up Python modules.

The observer pattern used by the framework naturally models event-based cameras and algorithms. However, this pattern can lead to the problem known as callback hell: deeply nested statements make the code hard to read. Languages such as Javascript have solved this problem with the async/await construct. This construct is available in C++, but is not compatible with the template deduction mechanism leveraged by the framework.

The current implementation of *partial event handlers* relies on *make* functions. These functions wrap the handlers constructors to enable template deduction. The C++17 standard allows template deduction from the constructor of a class, making the *make* functions unnecessary. The upcoming Debian 10 and macOS 10.15 operating systems will provide full support for this standard with their default libraries, allowing a major framework update.

The framework supports arbitrary event types and provides constructs to merge the output of several cameras. Thanks to both these features, it can be leveraged in the design of a three-chip event based camera to capture color events. The next chapter introduces a working prototype, and presents tracking algorithms that rely on color events.

Chapter III

Event-Based Color Segmentation With a High Dynamic Range Sensor

III.1 Introduction

Conventional computer vision approaches to color segmentation in videos can be organized in two categories: pre-deep learning and post-deep learning. The large number of pre-deep learning approaches [18] shows that they do not solve the problem generically. On the other hand, post-deep learning algorithms segment video with generic structures [141], however they require tremendous amounts of data and computational power, making them impractical on embedded systems. Moreover, the algorithms from both categories were originally developed on static frames, then extended to videos.

The benefits of event-based cameras make them well-suited candidates to overcome existing limitations in automated color segmentation: they are power efficient, and they generate sparse events which bundle spatial and temporal information. This property makes it possible to solve computer vision problems such as stereo-vision [159, 160], optical flow [161] or tracking [162] in an efficient and robust manner. Moreover, the cameras' high temporal resolution allows for simplifying assumptions, with complex behaviors emerging from simple, high-speed algorithms.

Most event-based cameras generate polarity events mimicking spikes in the ON and OFF visual system pathways. These events do not encode absolute luminance information. The luminance or color measurements of the DAVIS are performed in a frame-based fashion, therefore they do not benefit from the event-based approach's advantages. By contrast, the ATIS [163] provides event-based absolute luminance measurements, thanks to pixels that combine an asynchronous change detection circuit and a separate exposure measurement circuit. This chapter presents a functional event-based color sensor assembled from three ATIS, illustrated figure III.1, and its application for segmenting colored objects with simple processing techniques requiring little computation power. The absolute luminance information allows for a robust and computationally cheap color segmentation based on clustering, unlike change detectors which need to rely on edges detection. We evaluate the sensor's ability to track colored shapes, using a real-time on-line algorithm. Thanks to the nature of the data generated by event-based cameras, tracking can be implemented with a moving mean algorithm [164], which requires very little

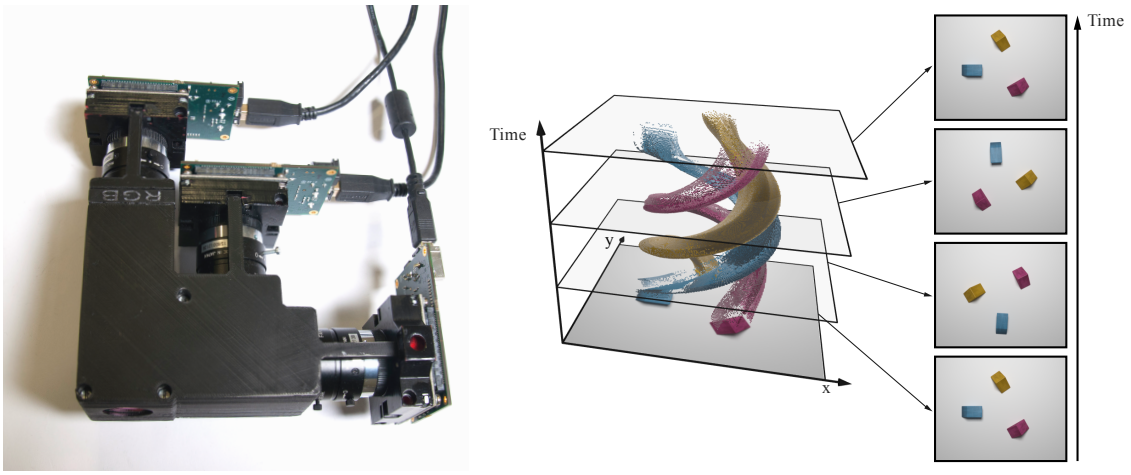


Fig. III.1 *The three-chip event-based camera is an assembly of three ATIS cameras (left). The cameras share the same field of view. Reconstructed color events can be visualized as a spatio-temporal point-cloud (right). There are as many color points in the four frames (far right) as in the whole point cloud.*

computational power. More complex and robust methods have been devised [16, 165] for more demanding applications.

III.2 Material & Methods

III.2.1 Three-chip event-based camera

We build an event-based color sensor as an association of three ATIS cameras acquiring red, green and blue light exposures. The sensor captures light through a hot mirror reflecting infra-red light. A beam splitter directs photons with wavelengths larger than 605 nm towards the red sensor. The other photons are reflected towards a second beam splitter, which directs photons with wavelengths smaller than 505 nm towards the blue sensor. The remaining photons are directed towards the green sensor. Before hitting the red, green and blue sensors, photons cross band-pass filters which mimic the filtering functions of conventional Bayer matrices' pixels. Each sensor uses a C-mount objective, as the sensors dimensions prevent using a common objective placed behind the hot-mirror. Figure III.2 illustrates the assembly.

III.2.2 Color events

In order to account for the mechanical imperfections of the prototype, a spatial calibration step is required to make sure that the color sensor's cameras share the same field of view. We capture a checker board with the sensor before each recording, and compute the homography linking the green and blue cameras to the red one. The homography is computed by determining the direct linear transformation on normalized points [166], which were extracted from a reconstructed image of the checker board using corner detection and structure recovery [167]. This spatial calibration is valid only for objects within

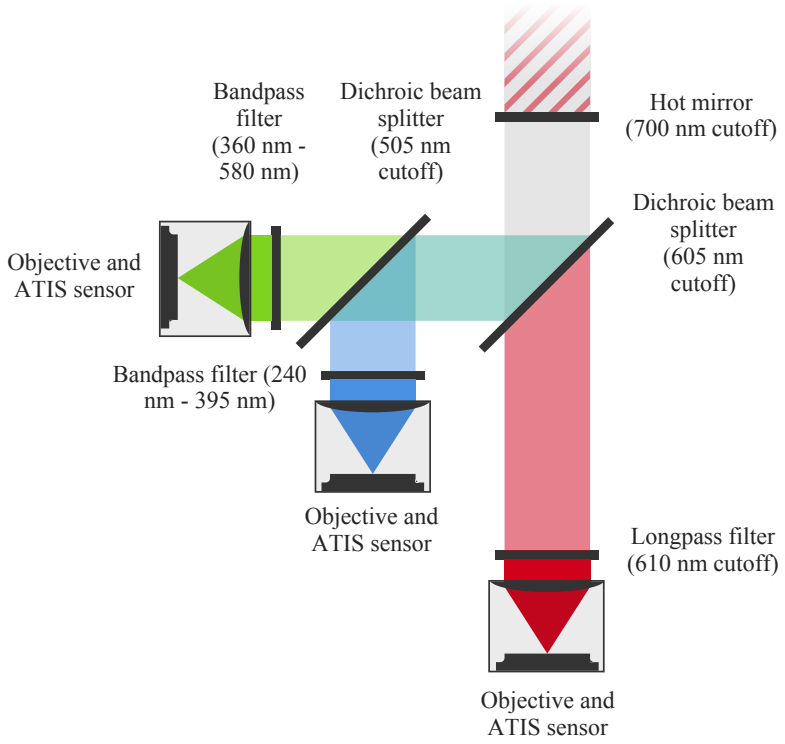


Fig. III.2 *Our three-chip event-based camera uses dichroic filters to split the light beam and ATIS cameras to record the scene as events. We use an objective for each camera instead of a single one in order to reduce the flange distance (constrained by the sensors' size).*

the checker board's plan. However, we observe a good pixel matching for objects in other plans as well, as the fields of view differences are small compared to the pixels size.

Therefore, the red sensor's pixel with index i has the same field of view as the green and blue sensors' pixels with index i. We call *pixel with index i of the color sensor* the virtual pixel combining the red, green and blue pixels with index i. The signal captured by this pixel can be modeled as a continuous $\mathbb{R}^3$ function s_i of the time t:

$$\begin{aligned} s_i : \mathbb{R} &\to \mathbb{R}^3 \\ t &\mapsto (r, g, b) \end{aligned} \tag{III.1}$$

where r, g and b are the red, green and blue components intensities of the signal. i, the pixel's index, is in the range $[1, n]$, where n is the sensor's number of pixels. We want the color sensor to generate events $e_{i,t}$ defined by the tuple of attributes:

$$e_{i,t} = (i, t, r, g, b) \tag{III.2}$$

Assuming an initial value $s_i(t_0) = (r_0, g_0, b_0)$, the pixel with index i's first event should be generated at the time t_1 such that $s_i(t_1) = (r_1, g_1, b_1)$ and the distance in $\mathbb{R}^3$ between (r_0, g_0, b_0) and (r_1, g_1, b_1) is larger than a tunable threshold. The distance function should

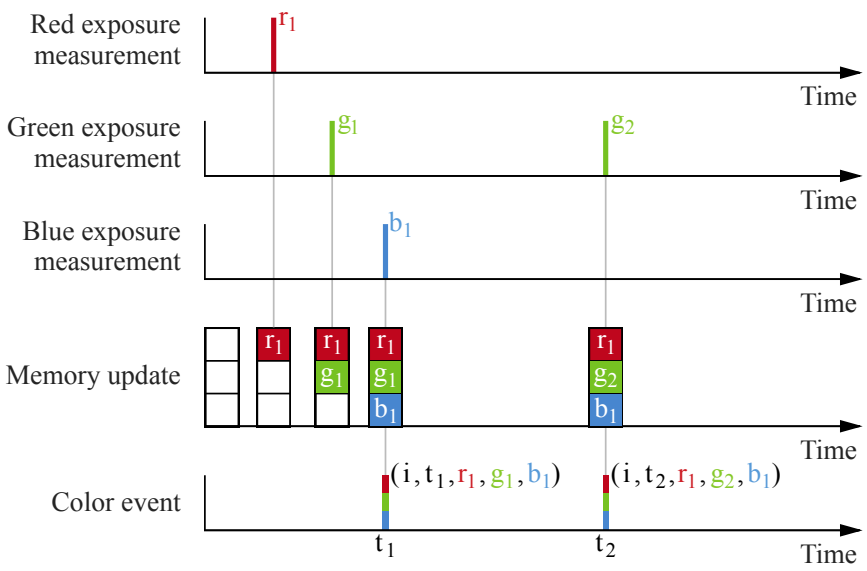


Fig. III.3 *Events generated by the color sensor's three cameras' pixel i are merged to generate color events. Every time an exposure measurement is generated by one of the ATIS cameras, it is stored in memory. Then, a color event is dispatched using the current memory value.*

not be the euclidean distance in order to mimic human perception, which is highly non-linear in RGB space [168].

The ATIS-based three-chip camera's pixels do not yield the s_i signal directly. Instead, the camera associated with each color component generates an independent stream of events. Since ATIS cameras yield exposure measurements with a delay inversely proportional to the measured exposure, it is not possible to detect temporally coinciding events to generate color events. Therefore, we associate each color sensor's virtual pixel with a memory space storing the three color components. Every time an event is generated by one of the color component cameras, the memory is updated and a color event based on the current memory value is dispatched. This mechanism is illustrated figure III.3.

III.2.3 Color model

We consider color object tracking as a first practical application of the event-based color sensor. Given a pre-determined set of uniformly colored objects, we want to determine the position of each object in an scene at every moment. We use a two-step approach : first, we build a statistical color signature for each object using a labelled scene. Then, events from an unknown scene are matched against the statistical models and associated with the closest signature.

In order to reduce the required amount of computation for each event, we reduce the problem's dimensionality by converting color events from the RGB space to the CIEL*a*b* space. We use only the a* and b* components of the latter. For each object, we gather events from the labelled scene and project them to the a*b* plane of the CIEL*a*b* space. We use a bivariate normal distribution as a statistical model for

describing these points.

Converting events from the RGB space to the CIEL*a*b* space requires a color calibration step. ATIS cameras send exposures as a pair of threshold-crossing events to the computer. The actual exposure is - as a first approximation - proportional to the inverse of the time difference between the two thresholds-crossing events. We use a Macbeth ColorChecker to evaluate the required proportionality factor between the time difference inverse and red, green and blue components in RGB space. We observe that the expected red, green and blue values given by the Macbeth ColorChecker as functions of the measured inverse threshold-crossing time differences are well described by affine functions, as shown figure III.4. The need for affine functions instead of linear functions can be attributed to the sensor imperfections, including the pixels' dark current. We calculate the affine regression by minimizing the mean squared error for each color component. This method yields good results for displaying the sensor's measurements using an RGB screen. Estimated red, green and blue components can be used to determine the CIEL*a*b* color components using several non-linear relations [169].

However, when using this model to convert the measured colors to the CIEL*a*b* space, we observe a poor fit with the values given by the Macbeth ColorChecker. The difference can be attributed to the uncorrelated regression applied to each component and the ATIS cameras' noise. Therefore, we use the Nelder-Mead simplex algorithm [170] to optimize the six parameters of the three color components' affine regressions. We minimize the distances between the expected colors given by the Macbeth ColorChecker and the measured points in CIEL*a*b* space. Since the CIEL*a*b* space is perceptually uniform, this method yields the best compromise for converting the measured Macbeth ColorChecker's colors to the CIEL*a*b* space with regards to human perception. Figure III.4 shows the two methods results.

III.2.4 Signatures

We consider the sequence S of n color events associated with a uniformly colored object:

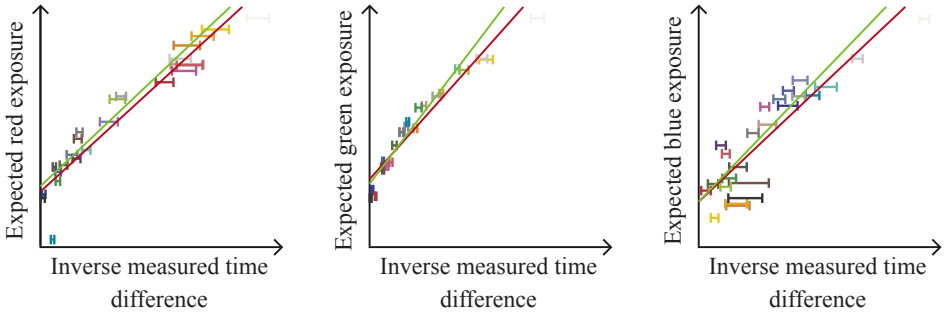
$$S = \left((i_k, t_k, r_k, g_k, b_k) , k \in [\![0, n-1]\!] \right) \tag{III.3}$$

We define S_{ab} as the sequence of pairs (a, b) obtained by converting each color event from the sequence S to the CIEL*a*b* space:

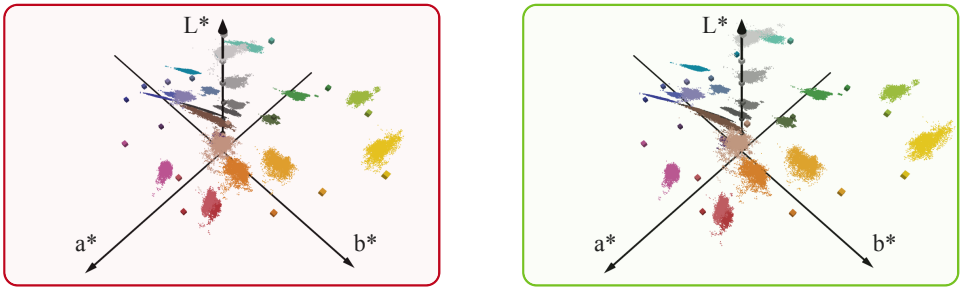
$$S_{ab} = \left((a_k, b_k) , k \in [\![0, n-1]\!] \right) \tag{III.4}$$

We call *signature* of the considered object the bivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ estimated from the S_{ab} sequence:

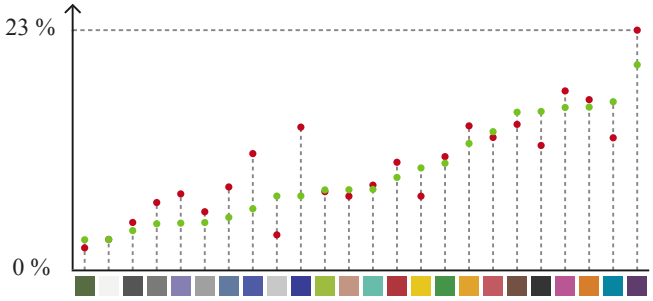
$$\begin{aligned} \boldsymbol{\mu} &= \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} = \frac{1}{n} \sum_{k=0}^{n-1} \begin{pmatrix} a_k \\ b_k \end{pmatrix} \\ \boldsymbol{\Sigma} &= \begin{pmatrix} \sigma_a^2 & \sigma_{ab} \\ \sigma_{ab} & \sigma_b^2 \end{pmatrix} \\ &= \frac{1}{n-1} \sum_{k=0}^{n-1} \left(\begin{pmatrix} a_k \\ b_k \end{pmatrix} - \boldsymbol{\mu} \right) \left(\begin{pmatrix} a_k \\ b_k \end{pmatrix} - \boldsymbol{\mu} \right)^{\mathsf{T}} \end{aligned} \tag{III.5}$$



Measurements of a ColorChecker's tiles are plotted against the expected RGB components. Affine functions appear as a good fit to model the transformations. We consider two methods to determine the functions' parameters : mean-squared errors minimization (red) and mean CIEL*a*b* distances minimization (green).



Points represent the sensor's measurements converted to calibrated RGB colors with each model, then converted to the CIEL*a*b* space. Cubes represent the tiles' expected colors' positions given by the ColorChecker.



The mean distance of the measured points to their expected position in the CIEL*a*b* space as a fraction of the largest distance between two colors is plotted for each tile.

Fig. III.4 *Converting the sensor's color events to the CIEL*a*b* space requires a transformation model. We use a Macbeth ColorChecker to compare the expected color values with the measured time differences. We consider two strategies for finding our model's parameters : mean squared error (red) and non-linear optimisation to minimise the distance between expected and measured colors in the CIEL*a*b* space (green). The latter yields a better fit for blue and purple colors in the a*b* plane. On the a*b* plane figures, squares represent the expected colors and circles represent the mean measured value. The represented Macbeth ColorCheckers were captured by our sensor.*

The experiment presented figure III.5 illustrates the method to determine the signature of actual objects. Five colored wooden pieces placed on a white background are recorded. Even though the scene is static, the ATIS cameras' noise triggers exposure measurements which are converted to color events. We associate a pixel set - or mask - to each wooden piece. The color events generated by this pixel set make up the S sequence used to fit a signature. The color signature for the background is evaluated as well.

III.2.5 Tracking

After determining the five wooden pieces' signatures, we consider a scene with the same pieces moving. Let X be the continuous bivariate random variable which associates each color event with its a* and b* components:

$$\begin{aligned} \mathbf{X}\colon \left(\mathbb{N}, \mathbb{R}^+, \mathbb{R}^3\right) &\to \mathbb{R}^2 \\ e_{i,t} &\mapsto \begin{pmatrix} a \\ b \end{pmatrix} \end{aligned} \tag{III.6}$$

We note $\mathbf{x} = \begin{pmatrix} a \\ b \end{pmatrix}$ a color event's a* and b* components.

Writing O_j for the probabilistic event "the color event was generated by the object j", the Bayes' theorem yields the probability that the object j generated the considered color event:

$$P\left(O_j \mid \mathbf{X} = \mathbf{x}\right) = \frac{f_{O_j}\left(\mathbf{x}\right) P\left(O_j\right)}{\sum_{j=0}^{m-1} f_{O_k}\left(\mathbf{x}\right) P\left(O_k\right)} \tag{III.7}$$

f_{O_j} is the probability density function of the bivariate normal distribution associated with the object j, and m is the number of objects.

The color event is associated with the object with the largest probability to be the source of the event. Assuming an identical probability $P\left(O_j\right) = \frac{1}{m}$ for each object to generate an event (six in the wooden pieces example, background included), we simply need to find the index j maximizing $f_{O_j}\left(\mathbf{x}\right)$, given by:

$$f_{O_j}\left(\mathbf{x}\right) = \frac{1}{2\pi\left|\mathbf{\Sigma}_j\right|} e^{-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_j\right)^{\mathsf{T}} \mathbf{\Sigma}_j^{-1} \left(\mathbf{x}-\boldsymbol{\mu}_j\right)} \tag{III.8}$$

where $\boldsymbol{\mu}_j$ and $\mathbf{\Sigma}_j$ are the object j' signature' mean and covariance matrix.

In order to track the objects, we use a moving mean algorithm. Each object is given a center $\mathbf{p_j} = \begin{pmatrix} x_j \\ y_j \end{pmatrix}$, where x_j and y_j are the object's mean coordinates in the screen referential. When an event $e_{i,t}$ is generated, the mean associated with the object minimizing expression III.8 is updated. The new mean $\mathbf{p'_j}$ is calculated from the previous mean and the event as:

$$\mathbf{p'_j} = \lambda \mathbf{p_j} + \left(1 - \lambda\right) \mathbf{x_i} \tag{III.9}$$

where $\mathbf{x_i}$ is the coordinates in the screen referential of the pixel which generated the event. λ is an inertia parameter ranging from zero to one. λ is generally given a value close to

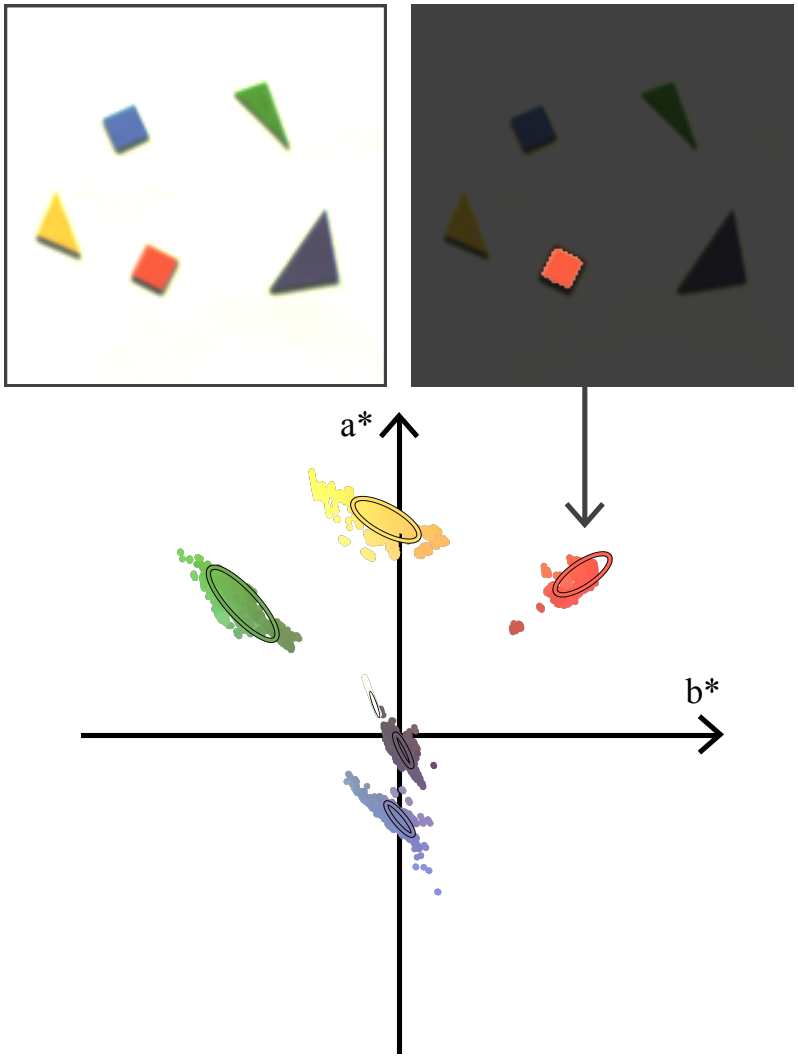


Fig. III.5 *In order to build color signatures for a set of wooden pieces, we accumulate noise-generated events from a static scene. We use the resulting image to build a mask labelling a specific piece (here, the orange one). All the color events associated with the mask's pixels are converted to the CIEL*a*b* space and projected on the a*b* plane. The projected points are used to estimate a bivariate normal distribution, which we call the signature. The bottom diagram shows 95% confidence ellipses of the wooden pieces' signatures in the a*b* plane.*

$(1 - 10^{-3})$. The larger λ, the more robust to noise the tracking. However, large λ values yield more latency and deteriorate the algorithm's ability to account for small variations.

We take into account the camera noise with a spatio-temporal activity filter. Once an event is associated with an object, we count the number of prior events associated with the same object that were generated less than one second before in a six-by-six square window around the event's position. Only events with at least thirty neighbors in this spatio-temporal window are taken into account for updating the object's mean position. Increasing the required count decreases the number of false positive events, while increasing the number of false negative events.

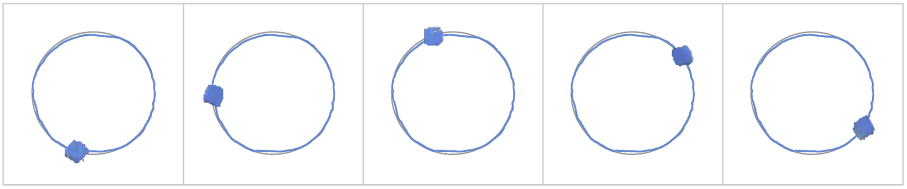
III.3 Results

We applied the color tracking algorithm to three experiments. Videos illustrating the associated results are provided as supplementary materials. The first experiment is recorded under laboratory-controlled conditions. Five colored wooden pieces are placed on a rotating surface captured with a static sensor. Figure III.6 shows the results compared to the ground truth, evaluated with a contour tracing algorithm [171]. The objects and their centers are identified on images reconstructed from the color camera's events. For each event, we calculate the distance between the associated object's estimated mean and its ground truth. The mean distance is given for each object as a fraction of the yellow object's trajectory's radius. We are able to estimate the objects trajectories using only color data. The moving mean algorithm's λ parameter is identical for all the objects, empirically set to $(1 - 10^{-3})$. A compromise must be reached between noise robustness and accuracy. Reducing the parameter's value would improve results for the purple object while degrading the results for the green one.

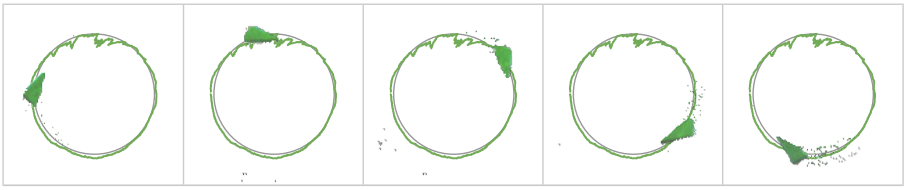
The second experiment consists of a moving camera in an urban scene containing a red road sign and a green pharmacy neon light. The corresponding color signatures are learnt from an initialization step. The latter uses data from a one-second sequence taking place before the experiment's recording. Figure III.7 illustrates the associated color reconstruction process. Due to the scene's high dynamic range, the linear color model yields little detail in the dark areas of reconstructed frames. Therefore, we generate color frames for display purposes by applying a logarithmic tone-mapping on each channel independently. This operation yields incorrect colors, but shows much more detail in dark areas. The tracking algorithm is not influenced by this operation since it uses colors calculated by the linear model, as presented in our methodology.

The third experiment takes place in an urban scene as well. It consists of two pedestrian wearing colored sweaters walking in front of the sensor. Figure III.8 shows the segmented events for both urban experiments, while figure III.9 compares the estimated trajectories with the ground truth. The latter is computed with the contour tracing algorithm provided by S. Suzuki et al. [171] as well. We remind the reader that this technique exploits shape rather than color: spatial constrains yield more robust results, but require a more complex algorithm. The estimated mean position lags behind the ground truth, which is a consequence of the moving mean algorithm. In order to assess the dynamics of our results, we compensate the lag for the road signs experiment and shift the position's reference for the pedestrians one. Table III.1 summarizes the mean errors and standard

Images reconstructed from the color camera' events over time

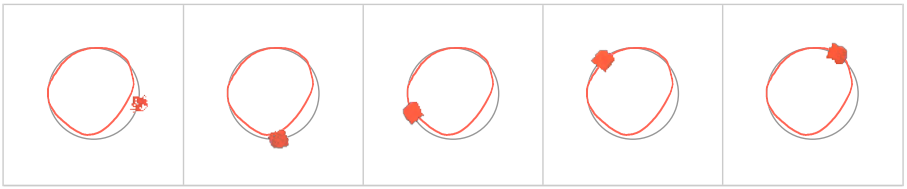


Mean error on blue position estimation: 1.6 %

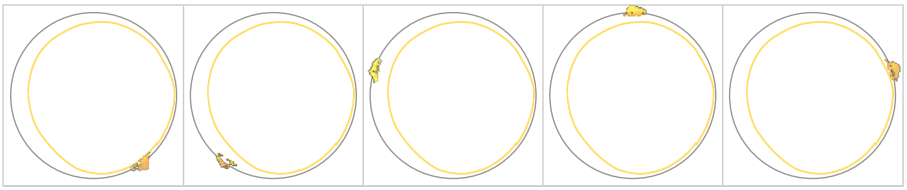


Mean error on green position estimation: 3.3 %

Mean error on purple position estimation: 10 %



Mean error on orange position estimation: 4.7 %



Mean error on yellow position estimation: 19 %

Fig. III.6 *Tracking of five wooden pieces in rotation motion. Figures show the pieces motions estimated with our method (colored trajectories) and the ground truth (grey trajectories) for a whole rotation. The mean error for each object is the average distance between the estimated object's mean position and the ground truth as a ratio of the yellow object's trajectory's radius. The observed errors derive from the compromise between noise robustness and accuracy imposed by the moving mean algorithm.*

Fig. III.7 *The figure's top row shows reconstructed frames from events acquired by each sensor of the three-chip event-based camera. The grey levels are tone-mapped with a logarithmic function in order to be displayed on a regular screen. The bottom-left frames are reconstructed with the linear color model presented in the methodology. The colors are properly reconstructed, but very little detail is left in dark areas. The bottom-right frames use the channels' logarithmic tone-mappings as their color channel, yielding incorrect colors but much more details in dark areas.*

deviations along the x and y axis for both urban scenes. We observe degraded performances for objects near the sensor's edges, which can be attributed to sensor limits. On the one hand, the ATIS camera used in the assembly lacks sensitivity to blue wavelengths, which is reflected by longer integration times for this component. This leads to timestamp differences between channels, which result in incorrect color reconstructions. On the other hand, our prototype three-chip color camera exhibits optical aberrations which degrade the signal near the edges.

III.4 Discussion

The event-based three-chip color camera is the first working prototype of an event-based sensor able to acquire absolute color information: the sensor generates packets of data carrying the luminance value integrated over a small time interval. By contrast, the event-based color pixel designed six years ago [73] and the DVS camera with a Bayer matrix built in early 2017 [172] can only detect color variations: they send the same message regardless the variation magnitude, and require heavy calculations to retrieve the absolute luminance. Even though our prototype is still at an early stage, we manage to track colored objects in several scenes, using only the color information generated by

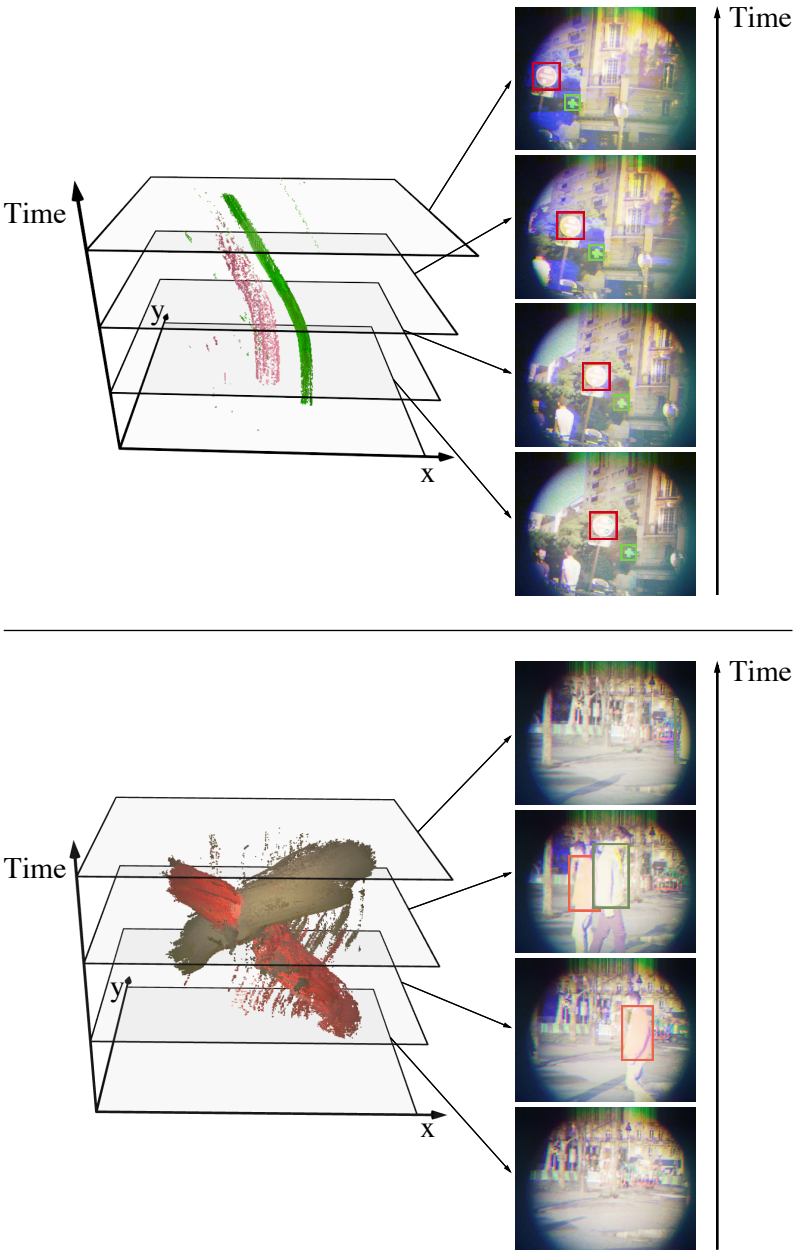


Fig. III.8 *We consider two outdoors scenes to assess the tracking algorithm performance: a moving camera acquiring a red road sign and a green pharmacy sign (top), and a static camera recording pedestrians wearing colored sweaters (bottom). The color signatures for the objects are calculated from similar scenes. The point clouds show color events where* f_O *is larger than* 10^{-5} *for one of the objects. These events are used to update the estimated center of the associated object with a moving mean algorithm where* $\lambda = 1 - 10^{-3}$*. The tracked object are framed on the reconstructed frames for better visualization.*

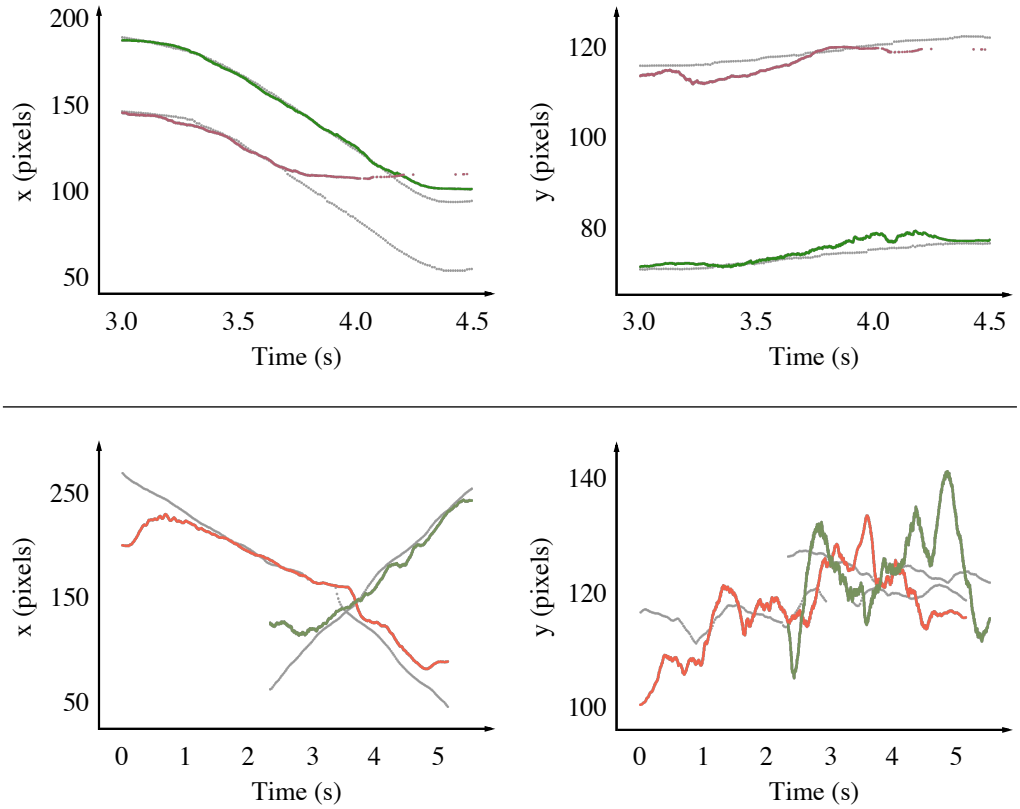


Fig. III.9 *We compare the estimated position of the tracked objects with the ground truth along the x and y axis as functions of time for the road signs experiment (top) and the pedestrians experiment (bottom). The events' timestamps are corrected to account for the delay induced by the mean-shift algorithm, effectively comparing dynamics rather than absolute values. Tracking is degraded near the edges, especially for the red stop sign (top, x axis, after* 3.6 *seconds), which is a consequence of our prototype's optical aberrations. Since the pedestrians move along the x axis, motion along the y axis is relatively small, making the noise appear stronger.*

Object	Mean error (pixels)	
	x	y
Green sign	2.18	1.30
Red sign	18.4	2.09
Orange sweater	14.5	4.58
Brown sweater	11.9	6.20

Table III.1: The mean error between the estimated position and the ground truth is evaluated for each tracked object in the outdoor scenes. The larger errors along the x axis for three out of four object can be attributed to the optical aberration near the sensors' edges.

the sensor. Thanks to the capture of absolute luminance, very little computational power is required to label the events. Therefore, the algorithm is a good candidate for the first stage of a complex chain of computations achieving a higher level task. It proves that color information alone is enough to achieve tracking with event-based cameras. The advantages of the event-based color sensor presented in this work over frame-based color cameras are similar to the advantages of grey event-based sensors over grey frame-based sensors: carrying out part of the computation on the sensor yields a natural data compression, with an increased temporal resolution. Both greatly reduce the required amount of computation for the processor. However, a proper use of the generated data requires a re-design of most computer vision algorithms.

The presented prototype can find applications in embedded systems. When low latency and low power consumption are required - as an example, with drones - conventional vision sensors show limits which can be overcome with event-based cameras. Fast color segmentation on a drone can be useful for several tasks, such as target detection and tracking or environment mapping. Moreover, the high dynamic range of the ATIS camera tackles the luminance adaptation issue, particularly troublesome for self-driving cars. Color makes road sign segmentation and recognition much easier on such systems.

The use of spatial information is out of the scope of this work. However, it should allow for a more robust algorithm thanks to data fusion, and is considered as this work continuation. We also identify several areas of improvement for the sensor that would benefit the algorithm's results. These improvements require hardware development. On the one hand, one of the event-based three-chip color sensor's weaknesses is its lack of sensitivity to short wavelengths. This limitation is shared by most silicon-based photodetectors, but is aggravated by the ATIS's low sensitivity to low light. Therefore, better results would be achieved with increased sensitivity. On the other hand, the three-chip architecture raises several issues: the resulting sensor is cumbersome, the chips must be temporally synchronized, and a custom optical system is required to align the fields of view, causing distortions and reducing the optical throughput. Having a single array of pixels would address all these issues, but requires designing a chip from the ground up.

Assuming the design of a new chip, it would be interesting to consider the following problem. Both the sensor presented in this work and the existing event-based color sensors digitize the analog light signal into events for each channel independently. Processing is then performed on the generated events, including color merging. The parallel drawn with the human eye for such sensors [163] ignores part of the eye complexity, including data passed between pixels through the horizontal cells. This data appears to be analog rather than digital. Implementing such a data transfer in the next generation of color neuromorphic vision sensors may be the key to acquiring color information efficiently. It may also help overcoming the following paradox in computer vision: for segmenting natural scenes, color, though helpful, provides a generally small advantage [173]. However, it requires dealing with three times as much data. Assuming an access to the extra power required to deal with this data, one is generally better off with a more complex algorithm working on grey levels. Merging colors on the analog level may help reducing the amount of generated data without tainting its quality.

The methods developed to explore event-based color vision can benefit systems beyond the scope of cameras, for they make it possible to manipulate arbitrary event types and communicate with multiple sensors. The field of visual psychophysics, which aims

to quantify the relationship between stimulus and perception, relies on a wide range of sensors: eye trackers, electroencephalogram and response buttons are a few examples. These sensors' output can be represented as precisely timestamped events, thus a psychophysics platform can be seen as an assembly of event-based sensors. Moreover, novel psychophysics results can serve neuromorphic engineering, as the latter draws inspiration from our understanding of biological systems. The next chapter presents a setup designed to study the role and importance of precise timing in primates' visual system.

Chapter IV

An event-based setup for high-speed visual psychophysics

IV.1 Introduction

Several recent neuroscience and psychophysics works suggest than the human visual system operates with a much higher temporal precision than once thought. Notably, humans are sensitive to flicker at 500 Hz [174], spikes in the optical nerve have a millisecond temporal accuracy [175], and eye microsaccades can be as short as 6 ms [176]. The possible role of precise timing in the human visual system - and, more generally, in the human brain - is critical to the design of neuromorphic devices and algorithms, notably since it is one of the most distinguishing features of event-based cameras.

High-framerate visual psychophysics experiments require a high-speed display device, precise sensors and precise synchronization. Such experiments have only recently become possible with off-the-shelf components, thanks to novel digital display technologies and eye trackers, both operating at frequencies in the order of the kHz. Nevertheless, easy-to-use software to run psychophysics experiments, for example *EventIDE* [177], are design for usual screens operating at 60 Hz or 120 Hz. Such software is executed on non-real-time operating systems, with the underlying assumption that the timing errors caused by the computer are negligible compared to the time precision required by the experiment. As a matter of fact, the timing errors reach a maximum of 5 ms (see subsection IV.2.1), whereas the time between two 60 Hz frames is 16.7 ms. However, such timing errors are large compared to the millisecond precision found in the optical nerve, and required for high-framerate psychophysics. This problem can be solved with the same approach used in event-based computer vision: smart sensors detect events in the signal they measure and precisely timestamp them, while high-level algorithms run on conventional computers. A framework with arbitrary event types and sensors fusion capabilities is required for this task, since each sensor taking part in the experiment will generate a very specific data type.

This chapter presents the application of our event-based framework to a high-framerate psychophysics platform. This resulting device is a $19 \times 24 \times 9$ cm box, shown figure IV.1, containing a 1440 Hz binary projector, an embedded computer and a dedicated microcontroller for high temporal resolution user inputs detection. A two-buttons response box

is associated with the system. Facilities are provided to synchronize and communicate with a 500 Hz eye tracker. The box's schematics and code are both open-source and freely available online. Off-the-shelf components are used, resulting in a low-cost and easy to replicate solution.

IV.1.1 Existing high-frame-rate displays

Several novel display technologies have been used for high-speed visual psychophysics. L. Sawides et al. [178] used Texas Instruments' DMD technology to display stimuli at 22 700 Hz, but limited to 100 frames long binary patterns at a time. The vision research projector ProPixx, also based on the DMD technology, allows continuous display of 1440 Hz greyscale patterns. However, this solution is significantly more expensive than the standalone chip. CH. Poth et al. [179] explored Nvidia's G-Sync technology as a means to present stimuli with high temporal resolution, below 1 ms. As far as frame-rate is concerned, this approach is limited by a minimum presentation duration imposed by the monitor. Only presentations lasting for at least the minimum duration (at best 4 ms, depending on the monitor), can then be controlled with a sub-millisecond precision.

IV.1.2 Similar setups

Psychophysics setups based on DMD devices [180, 181] effectively result in open and low-cost ProPixx-like solutions. Nevertheless, these setups require both time and expertise to be replicated and validated, and rely on hard-to-reproduce dedicated hardware. In contrast, 60 Hz experiments benefit from consumer monitors and accessible frameworks in multiple programming languages, including Python[1], JavaScript[2] and Matlab[3].

IV.1.3 Components

The box's projector is a *DLP Lightcrafter 3000*, developed by Texas Instruments. In order to circumvent the diamond-shaped pattern of this device, the projector is tilted by 45 degrees, resulting in a 343×342 display. Rather than using a real-time operating system, for which some hardware do not have support, the box is based on a pair of computers with specific roles. The first one, a *Jetson TX1* board, runs a non-time-deterministic operating system (Ubuntu 16.04 LTS) and manages heavy-calculation tasks, notably video decoding. The second computer is a *Teensy 3.5* board running a single program - ours - with interrupts to precisely (down to 1 µs in most cases) timestamp subject's inputs, DMD frames and eye tracker samples. This hybrid solution benefits from the many tools developed for these two specific use cases. We characterize the device temporal precision using an external photo-diode and oscilloscope.

[1] https://www.psychopy.org
[2] https://github.com/psychopy/psychojs
[3] http://psychtoolbox.org

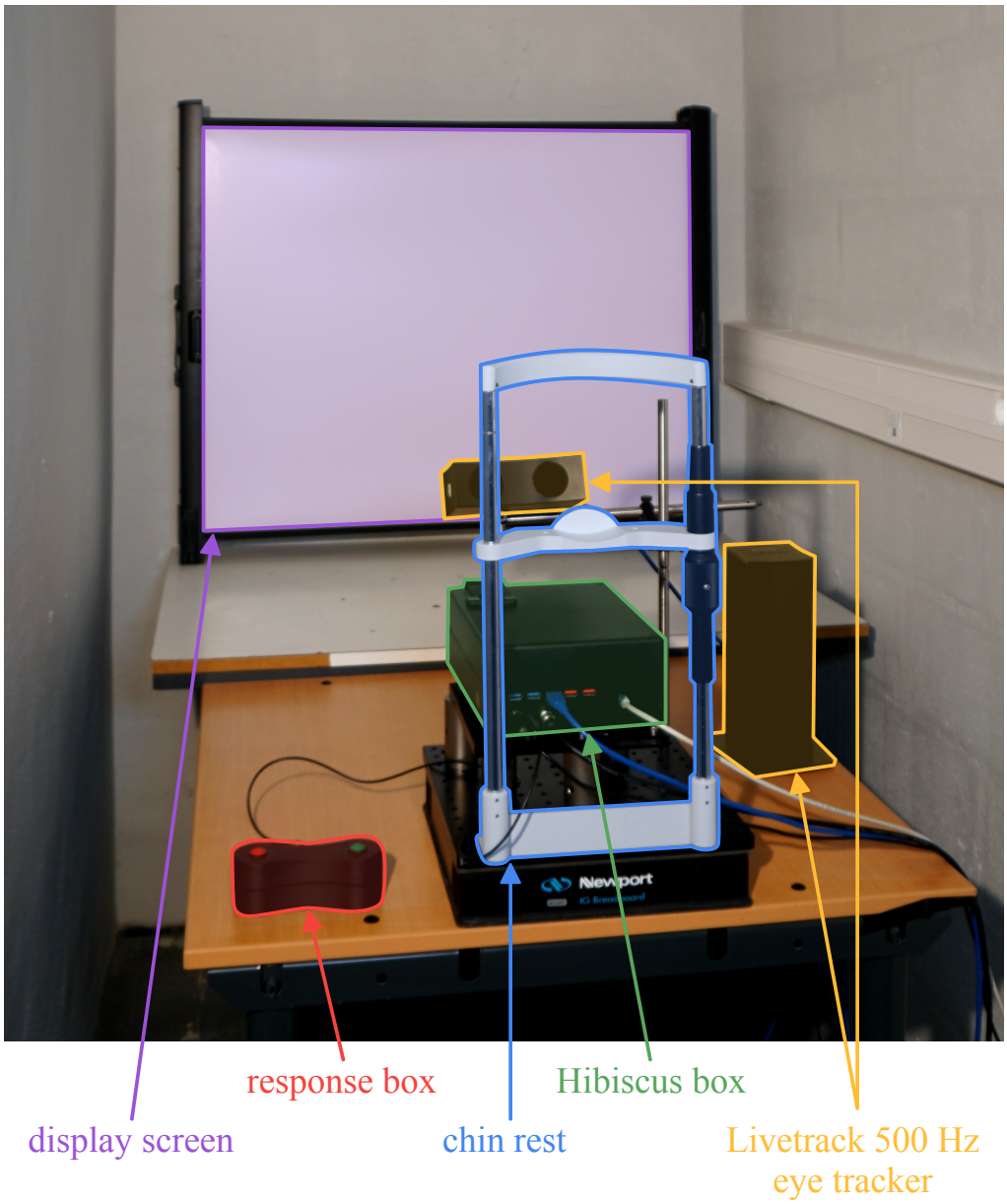


Fig. IV.1 *This figure shows the setup described and characterized in this work. The Hibiscus box, shown in green, contains a DLP Lightcrafter projector, a Jetson TX1 board and a Teensy 3.5 board. The response box, part of the presented solution, is shown in red. The complete setup makes use of a* 500 Hz *Livetrack eye tracker, held with optics assembly parts instead of the provided base as to not obscure the projector.*

background task	`sched_min_granularity_ns`	median delay	peak delay
none	8000 µs	81 µs	3993 µs
	800 µs	53 µs	3043 µs
`stress-ng -cpu 4`	8000 µs	76 µs	4042 µs
	800 µs	57 µs	3950 µs

Table IV.1: In order to estimate the real-time capabilities of the *Jetson TX1* board's pins, we perform the following experiment. A *Teensy 3.5* sends a signal to an input pin of the *Jetson TX1*. Upon detection by a program running on the *Jetson TX1*, an output pin connected back to the *Teensy 3.5* is triggered. The latter can measure the delay between the signals sent and received. The experiment is performed with and without a calculation-heacy task running in the background (namely, `stress-ng -cpu 4`). Two values are considered for the kernel's `sched_min_granularity_ns` parameter: the default 8000 µs and 800 µs. The program is run with a -20 niceness for each combination of conditions. A signal is sent every 100 ms, for 100 s. We observe no effect of the kernel parameter change, and, a reduced latency under stress. Non-negligible latency peaks appear with every set of conditions, preventing the use of the *Jetson TX1* as sole machine.

IV.2 Preliminary considerations

IV.2.1 Precise synchronization

Common operating systems (including Linux based operating systems, Windows and macOS) are not real-time, in the sense that a specific task is not guaranteed to complete in a determined amount of time. This behaviour is a feature: preventing CPU context switches increases overall throughput. However, when it comes to precisely timing user inputs, non real-time operating systems induce noise, as the task responsible for measuring time when an input is detected may be delayed by the kernel. The actual delay can reach several milliseconds [182], which is not negligible in kilohertz psychophysics. We perform a similar experiment using a *Jetson TX1* board's pins. We reach similar conclusions regarding delays, as shown table IV.1.

Real-time operating systems, on the other hand, give strong guarantees regarding task completion. However, such technologies are generally proprietary: the only real-time operating system supporting the *Jetson TX1* is RedHawk Linux[4]. Other limits include reduced throughput and longer software development times.

We therefore elect a hybrid solution. The *Jetson TX1* is associated with a *Teensy 3.5* board responsible for timestamping user inputs. The latter's time precision is estimated in section IV.4. This solution provides real-time timestamping while relying on open, well-known systems with a broad community of users.

[4]A list of the distributions compatible with the Jetson is available at https://elinux.org/Jetson_TK1#Linux_distributions_running_on_Tegra, and details on the real-time operating system Redhawk can be found at https://www.concurrent-rt.com/products/redhawk-linux

IV.2.2 1440 Hz display

Digital Micromirror Device, or DMD, is a proprietary technology by Texas Instruments. It consists in an array of small mirrors tilted by micro-electrical-mechanical systems (MEMS). DMDs can be used as video projectors by shining a powerful light source on the mirrors. Each mirror will either redirect the light to a screen or to a light absorber, depending on its tilt angle.

Digital displays refresh rates used to be limited by the pixels' response time (several milliseconds for liquid-crystal displays). Novel pixel types, such as micromirrors or light-emitting diodes, have much smaller response times: 16 µsand a few tens of nanoseconds, respectively. Given this tremendous improvement, the limiting factor becomes the bandwidth of the cable used to carry video data from the computer to the digital display.

The *DLP Lightcrafter 3000*, developed by Texas Instruments, is built upon a 608×684 mirrors DMD. It features a 1440 Hz binary pattern display mode. The limited bandwidth problem is circumvented as follows: the computer hosting the video sends 60 Hz RGB888 frames over HDMI, as it would with a usual screen, however each frame's 3×8 bit planes are interpreted as a moving binary pattern rather than color depths (hence the $3 \times 8 \times 24 = 1440$ Hz frame-rate). Even though any DMD projector has the theoretical ability to switch its mirrors at 1440 Hz, the *DLP Lightcrafter 3000* is, to our knowledge, the least expensive light projector providing this feature without requiring the development of custom electronics. A similar driving principle could be applied to LED displays. To our knowledge, however, such devices are not yet available as off-the-shelf components.

The *DLP Lightcrafter 3000*'s pixels follow a diamond pattern, which can be visualized figure IV.2. This pixel arrangement may have an impact on perception, making the comparison with experiments using square pixels more complex. Therefore, we tilt the projector at a 45 degrees angle counter-clockwise along its axis, revealing a rectangular array of 343×342 pixels. The figure IV.2 shows the portion of the projector actually used by our method, as well as the image sent by the computer over HDMI, which accounts for the diamond pattern's coordinate scheme. The pixel of the 343×342 frame with coordinates (x, y) (where $(0, 0)$ is the top left corner) must have, in the 608×684 frame sent by the computer, the coordinates (x', y') are given by $x' = 133 + \left\lfloor \frac{x+y}{2} \right\rfloor$ and $y' = 342 - x + y$, where $\lfloor\ \rfloor$ is the floor function. The x offset, 133, is chosen as to center the tilted area. Similar equations can be derived for other shape factors. The chosen shape factor maximizes the number of pixels inside the tilted frame.

The 1440 Hz frames projected by the *DLP Lightcrafter 3000* have uneven, but deterministic, durations. They follow a pattern repeated every 24 frames, with the following properties:

- the frames 1 to 22 and 24 of each pattern last $\frac{\lambda}{1440}$ s, instead of the expected $\frac{1}{1440}$ s duration
- the frame 23 (second before last) of each pattern lasts $\frac{24-23\lambda}{1440}$

We estimate $\lambda = 0.979$ from a measured 23^{rd} frame duration of 1030 µs. If deemed relevant, these variations can be accounted for in the psychophysics experiments performed with the device.

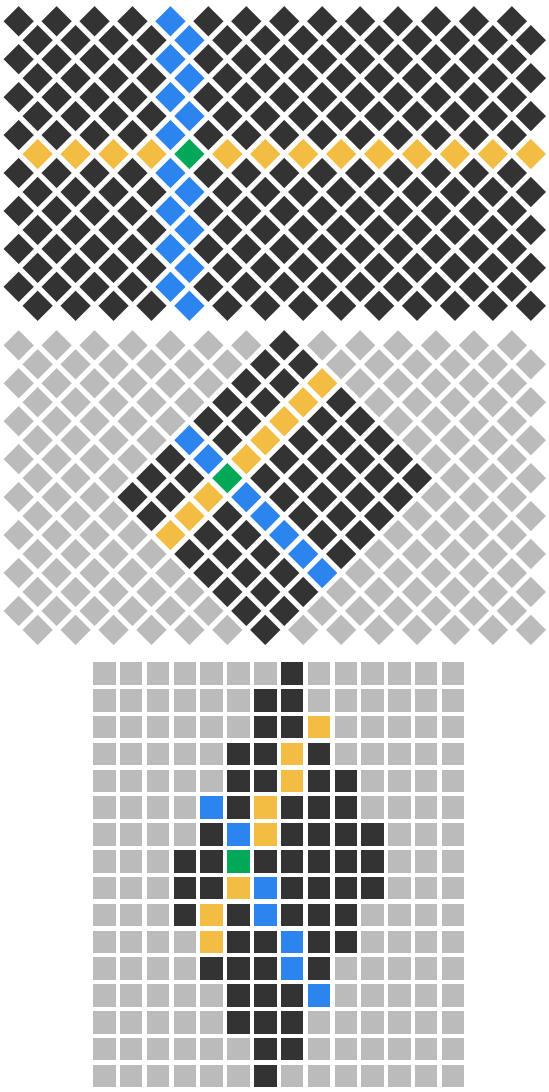


Fig. IV.2 *The DLP Lightcrafter 3000's pixels are arranged in a diamond pattern, shown by the top picture (on a smaller array for readability purposes). The colored pixels illustrate the coordinate scheme: the blue pixels' x coordinate is 5, and the yellow pixels' y coordinate is 7. The top left pixel has coordinates* $(0, 0)$*, and the green pixel has coordinates* $(5, 7)$*. The middle picture shows the rotated area actually used by our method, which has square pixels. The green pixel has coordinates* $(3, 2)$ *in the rotated frame reference. The bottom picture shows the rotated area inside the frame sent by the computer. The distortion is a result of the DLP's diamond pattern coordinate scheme.*

IV.2.3 Encoding stimuli

The method presented in this work assumes that stimuli are generated before the experiment and stored on a hard drive. The stored frames have $(608, 684)$ pixels, in order to match the format expected by the *DLP Lightcrafter 3000*. Alternatively, one could store frames with 343×342 pixels and calculate the expected frame at run time. However, this approach would require more processing power, and leads to a loss of generality, as a pipeline reading $(608, 684)$ frames can drive a non-tilted device.

Raw 608×684 RGB888 frames at 60 Hz occupy about 599 MB of memory for each second of video. Stimuli longer than a few seconds do not fit in most computers' RAM, and as such must be read from disk in real time. Video encoding algorithms are needed to meet this constraint.

Video encoding algorithms are generally lossy: they exploit the fact that well-chosen, irreversible changes in the original data can greatly reduce the encoded media size, while having little to no impact on the human perception of the decoded media. However, in our specific case, the data's bit planes sent to the projector do not encode colors, as assumed by encoding algorithms, but temporal patterns. As an example, changing a red channel byte with the value 01111111_2, representing the decimal value 127, to 10000000_2, representing the decimal value 128, does not have a huge perceptual impact. If this operation helps to reduce the overall encoded media size, it is an acceptable lossy irreversible change. Yet, the same byte interpreted as a binary pattern yields an almost-always on pixel in the first case, and an almost-always off pixel in the second. It follows that lossy algorithms cannot be used to encode the data expected by the *DLP Lightcrafter 3000* in 1440 Hz binary pattern mode, because they are likely to massively degrade the media.

State-of-the-art encoding algorithms include H.265 and VP9 [183], with lossy and lossless variants. However, hardware support for these algorithms remains scarce. As an example, the Jetson TX1 used in this work does not support hardware-accelerated decoding for either format. Instead, our processing pipeline relies on the H.264 encoding algorithm, which features lossless encoding and is hardware-accelerated on many platform, including the Jetson TX1. Note, however, that the Raspberry Pi, a smaller and less expensive embedded computer, does not provide hardware acceleration for lossless videos (though full HD lossy H.264 can be decoded in real time).

Despite being supported by the Jetson TX1, decoding lossless H.264 data is not straightforward for two reasons. First, H.264 expects Y*UV data (in contrast with RGB888 data). The conversion from RGB to Y*UV and back, though mathematically reversible, is not lossless for 8 bits integers. Secondly, the Jetson only support lossless decoding for chroma subsampled H.264 videos, which contain half as many bytes as their fully sampled counterparts for a given resolution (4:2:0 chroma subsampling). We solve both problems by writing (without conversion) each RGB888 frame into a twice as wide Y*UV420p frame. Unlike Y*UV420p, RGB888 channels are interleaved. We introduce a custom deinterleaving scheme: R and G channels are written over Y*, and the B channel is written over U and V (alternating with each line). This strategy is illustrated figure IV.3, and yields better compression relatively to direct memory copy of the interleaved RGB frame into the Y*UV one, as shown table IV.2. Moreover, it has a beneficial side-effect: the custom encoded Y*UV420p is close to the original video, even when decoded as if it were a standard Y*UV420p-encoded stream, allowing easy stimuli identification using a

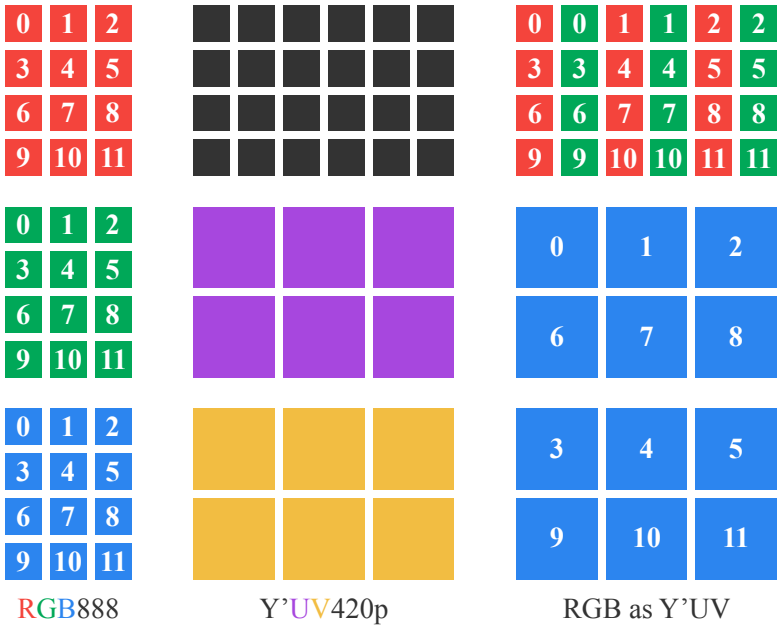


Fig. IV.3 *In order to lossessly compress RGB888 frames with H.264, we convert them to Y*UV420p. The left column shows the RGB888 bytes of a 3 × 4 pixels frame. The three channels are interleaved in memory. The middle column shows the Y*UV420p bytes of a 6 × 4 frame. The total number of bytes is identical for both encodings despite a different frame size, as Y*UV420p includes chroma subsampling. The right column shows the proposed method to write the RGB bytes inside the Y*UV frame.*

standard video player (despite erroneous colors, since the bit planes represent patterns and not colors). An example is given figure IV.4.

IV.3 The box and code

IV.3.1 The box assembly

A detail of the box's components is shown figure IV.5. The black ABS plastic box is a 24 × 19 × 9 cm electronic project box. It is 4 mm thick. The other mechanical parts are 3D-printed PLA. 3D models of these parts can be downloaded for free from the project's Github page. Assembly requires plastic cutting to create windows in the box for the focus wheel and the front ports. The components are held with M2 and M3 bolts. Visible 3D-printed parts are painted black to reduce their salience during psychophysics experiments. On top of easing transport and reducing electronics wear over time, the box occults the light emitted by the multiple blinking LEDs of the *Jetson TX1*, the *Teensy 3.5* and the *DLP Lightcrafter 3000*. The USB hub breakout board is extracted from an off-the-shelf USB hub. The custom 5 V power supply board features two *Austin MiniLynx* chips to power the *DLP Lightcrafter 3000* and the USB hub. Each *Austin MiniLynx* is associated with a potentiometer to configure its output tension. A third pair is visible figure IV.5,

frame rate	coherence	direct copy encoded size	interleaved encoded size	ratio
60 Hz	50 %	15.8 MB	4.4 MB	3.59
	30 %	17.9 MB	5.8 MB	3.09
	10 %	17.2 MB	5.9 MB	2.92
1440 Hz	50 %	18.5 MB	12.2 MB	1.52
	30 %	21.9 MB	13.3 MB	1.65
	10 %	26.1 MB	14.9 MB	1.75

Table IV.2: This table compares two possible lossless RGB888 to Y*UV420p conversions and their impact on H.264 encoding. Psychopy's DotStim function is used to generate stimuli with different frame-rates and coherence values, with constant speed and duration. The non-encoded stream is 374 MB large. The direct copy yields a Y*UV420p stream very different from usual videos, degrading the encoding algorithm performance when compared to our custom deinterleaving algorithm. The smaller the coherence, the more information is stored in the file (the video is harder to predict), hence the increasing encoded media size. Similarly, the higher the frame-rate, the farther the generated stream is to usual videos, since bit planes represent motion and not color.

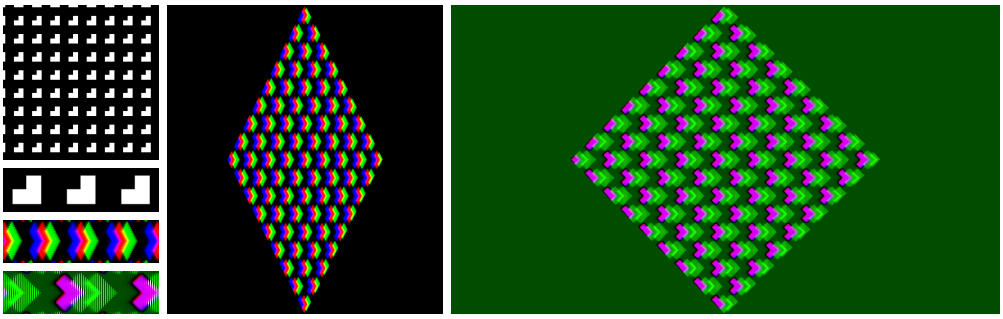


Fig. IV.4 *This figure illustrates the different transformations applied to the intended stimulus (top left square and its detail below) before H.264 compression. First,* 1440 Hz *frames are packed as the bit-planes of a* 60 Hz *color video (rectangle with a black background and its detail on the left, second from the bottom). The color channels are then interleaved, as shown figure IV.3, resulting in a twice-as-wide color video stream (rectangle with a green background, its detail on the left, first from the bottom). When opened with a conventional video player, the stimulus appears as seen on the right. Even though its spatial features are degraded and its colors are incorrect, it can help identify the stimulus file.*

added during assembly to accommodate for a potential extra device. Since all the pins of the Teensy 3.5 are interrupt-capable, the jack and BNC wires can be connected to any GPIO. The jack wires are pulled up to 5 V by 4.6 kΩ resistors.

IV.3.2 Synchronizing components

The *Teensy 3.5*'s clock is chosen as the reference to timestamp events. User inputs transmitted by the response box are directly detected as interrupts on the *Teensy 3.5* pins, as shown figure IV.6. Likewise, the *DLP Lightcrafter 3000*'s frames are precisely timestamped by the reference clock. Once timestamped, the events can be safely transmitted with non-time-deterministic protocols (USB and non-real-time operating system) to the *Jetson TX1* for further processing and logging. See section IV.4 for an estimation of the precision offered by the *Teensy 3.5*'s interrupts.

Detecting the DMD trigger's pulses is not sufficient to properly synchronize the displayed stimulus with user inputs. As a matter of fact, this trigger does not transmit a frame identifier, hence only the *Jetson TX1*, which drives the HDMI transmission, knows which frame is being displayed. Explicitly synchronizing the *Teensy 3.5* and *Jetson TX1* clocks, though possible using a time synchronization algorithm [184], is hard to validate using our specific hardware (since the *Jetson TX1*'s outputs are known to induce random delays, see IV.1). Instead, we exploit a defect of the *DLP Lightcrafter 3000*: the inter-frame time is slightly longer once every twenty-four 1440 Hz frames, on the boundary of every 60 Hz RGB888 frame read from HDMI. Upon detection of this longer frame, a USB message is sent to the *Jetson TX1*, which associates the label of the previous HDMI frame with the precise timestamp provided by the *Teensy 3.5*. This protocol, illustrated figure IV.7, can be checked for consistency at run-time, thus validating the synchronization and allowing errors detection. The two-frames offset which appears in the protocol accounts for the GPU double-buffering and the *DLP Lightcrafter 3000* behaviour[5], each inducing a one-frame delay. A quantification of the errors for a typical psychophysics experiment is given in section IV.4, with a consideration of its implications.

The *Livetrack* eye tracker communicates data over USB at 500 Hz, using its own clock to timestamp eye samples. The state of the Livetrack's digital inputs (including a BNC port) are provided with each sample. In order to estimate the Livetrack's samples times with respect to our reference clock (the *Teensy 3.5*'s), we periodically change the logical state of the BNC port. The switching times are driven and precisely timestamped by the *Teensy 3.5*, and reflected in the USB packets transmitted by the *Livetrack*. Upon detection by the *Jetson TX1*, a USB command is sent to the *Teensy 3.5* to trigger a new BNC logical change. This mechanism, illustrated figure IV.8 , makes it possible to associate each timestamped logical change with one - and only one - Livetrack sample. The whole cycle takes about 25 ms. Samples in-between BNC logical changes are timestamped using a linear regression based on the Livetrack's timestamps. Under the assumption that the BNC logical changes happen randomly during the Livetrack 2 ms inter-sample, the *Teensy 3.5* provided timestamps are matched with Livetrack timestamps offset by 1 ms on average. This is taken into account in the linear regression to provide an unbiased estimator.

[5] http://www.ti.com/lit/ug/dlpu006e/dlpu006e.pdf

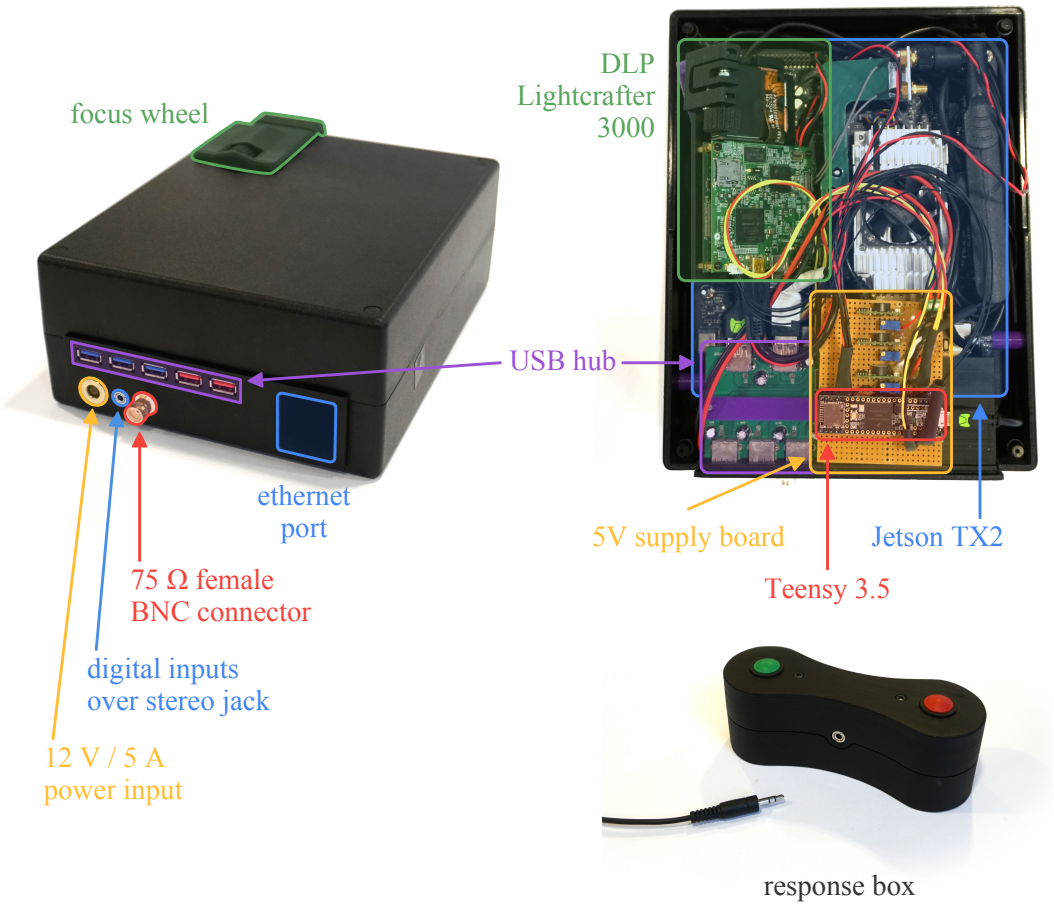


Fig. IV.5 *The Hibiscus assembly extends the Hummingbird assembly with a real-time micro-controller and an embedded GPU, providing an all-in-one device for displaying kilohertz stimuli and recording subjects' responses. The included Jetson TX1 development board (10) decodes H.264 frames displayed at* 1440 Hz *by the DMD Lightcrafter connected over HDMI (9). External events are timestamped with a microsecond precision thanks to the embedded Teensy 3.5 board (7). The front direct digital ports (2 and 3) make it possible to connect and synchronize external devices, such as a two-buttons remote for decision-making experiments, and a Livetrack eye tracker. The ethernet (4) and USB (5) hubs are connected to the Jetson, and are used for remote control and Livetrack communication, respectively. The Jetson board is directly powered by the input (1), whereas the other components use dedicated 5 V power regulators (8). The stereo jack's signal lines are connected to pull-up resistors and the Teensy's GPIOs. The focus wheel (6) provides a precise control over the Lightcrafter's focal length.*

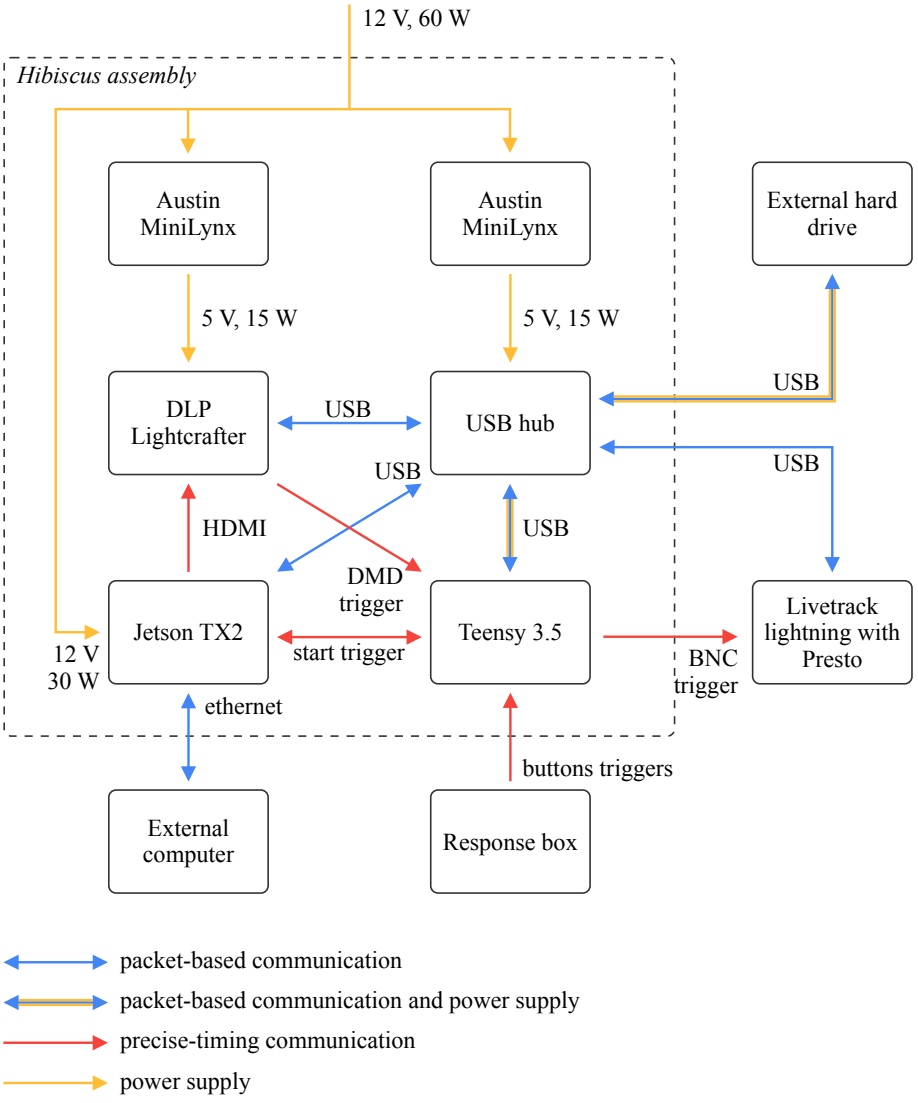


Fig. IV.6 *The Hibiscus box is powered by a single 12 V, 60 W source. The setup connections can be organized in two categories. The USB and ethernet connections (shown in blue), which have non-deterministic trip times, transmit non-time-precise information. Such data is timestamped before transmission by the reference clock (the Teensy 3.5 clock), allowing delayed transmission without precision loss. The transmissions shown in red feature synchronization mechanisms (HDMI), or are simple wires transmitting pulses (all the triggers). The start trigger is responsible for booting the Jetson TX1 on power up.*

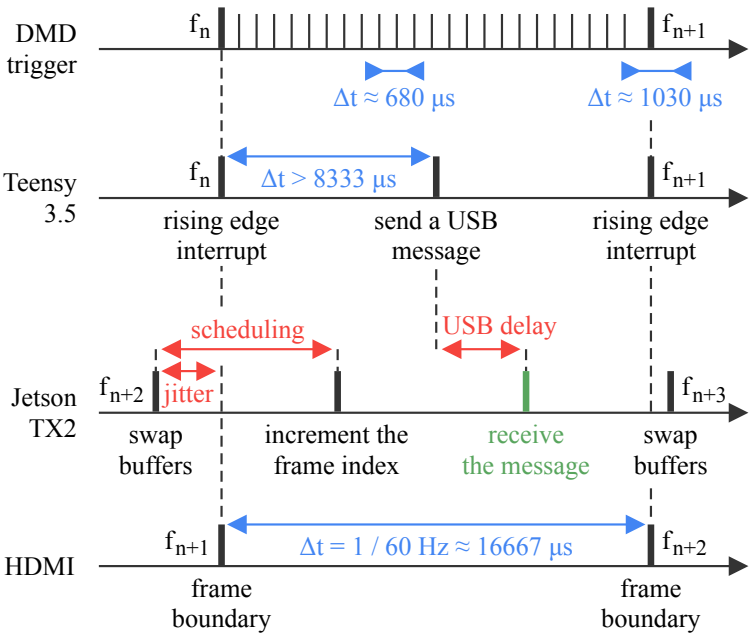


Fig. IV.7 *All the subject inputs and the frames timings are recorded by a single reference clock - the Teensy 3.5's clock. This dedicated micro-controller is responsible for precisely timestamping the inputs and sending them to the Jetson TX1 for further processing. The Teensy 3.5 does not know which frame is associated with a given DMD trigger rising edge. In order to avoid explicit clock synchronization, a labelling packet is sent to the Jetson TX1 at each* 60 Hz *frame boundary. Assuming short enough USB delays, precise timestamps can be associated with frame indices by the Jetson TX1. Both the Teensy 3.5 and Jetson TX1 keep track of the current frame index, allowing to detect and correct synchronization errors. This scheme requires at least one USB packet to be transmitted in less than* 8 ms. *This condition is actually met by most packets.*

IV.3.3 Code structure

All the code needed to use the box can be downloaded for free on the project's Github pages[6]. The code is organized in two modules. The first one, *Hummingbird*, can be used with a *DLP Lightcrafter 3000* and any computer. It provides tools to generate binary patterns and play them over HDMI at 1440 Hz. The second module, called *Hibiscus*, is built on top of Hummingbird. It supplements the latter with tools to run psychophysics experiments, and implements the synchronization mechanisms presented in section IV.3.2 (including the communication with the *Livetrack* and the *Teensy 3.5*). An open-source implementation of the Livetrack n-point calibration algorithm, compatible with the tilted display, is provided as well. Both modules expose command-line programs to use their features, and header-only C++ code to include them in other projects.

[6] https://github.com/neuromorphic-paris/hibiscus and https://github.com/neuromorphic-paris/hummingbird

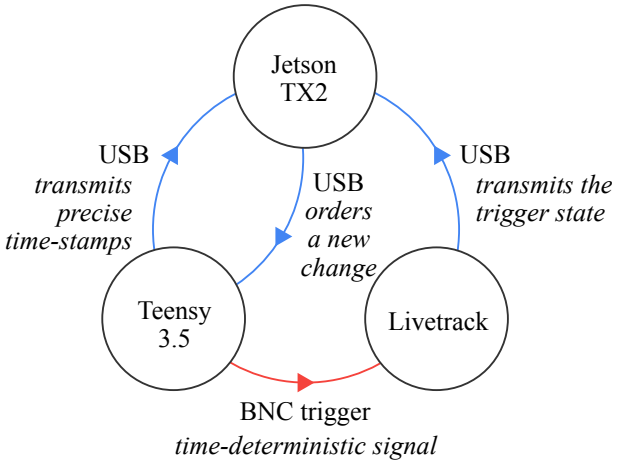


Fig. IV.8 *The Livetrack uses its own clock to timestamp data. In order to convert these timestamps to the Teensy 3.5's clock referential, messages are exchange in a cycle. The Teensy 3.5 uses its clock to precisely measure time and changes at the same time the logical level of the BNC trigger. The precise timestamp is sent to the Jetson TX1. The Livetrack provides samples of the trigger synchronously with eye-tracking data, at* 500 Hz. *The Jetson TX1 detects changes between consecutive data reports, and receives timestamps from the Teensy 3.5. When both information have been received, the Livetrack timestamp can be associated with a Teensy 3.5 timestamp. The Jetson TX1 completes the cycle by requiring a new change from the Teensy 3.5. Changes happen approximately every* 25 ms. *Livetrack samples between matched timestamps are converted to the Teensy 3.5's clock referential using a linear regression.*

a) Generating stimuli

The figure IV.9 gives an overview of the functions used to generate a 1440 Hz stimulus. The listed *.hpp* files reside in the Hummingbird git repository. *rotate.hpp* implements the rotation presented in section IV.2.3 and *deinterleave.hpp* the strategy for lossless H.264 compression from the same section. The *hummingbird.py* python module implements the same functions using numpy, and can be integrated in a Psychopy script, to use the latter for stimuli generation. Both the C++ and python programs rely on the external tool FFmpeg [185] for the toolchain's most complex step, namely encoding the Y*UV420p stream to H.264. The *generate* toolchain output is an H.264 encoded MP4 file ready for use by the *play* toolchain.

b) Playing stimuli

The figure IV.10 illustrates the functions used to play a stimulus encoded by the *generate* toolchain. *decoder.hpp* relies on the Gstreamer library [186], which provide hardware-accelerated and software implementations of the H.264 decoding algorithm. Thanks to this library, the toolchain is efficient when hosted by the *Jetson TX1* while being compatible with most desktop computers, regardless their hardware. The libav library [187] provides

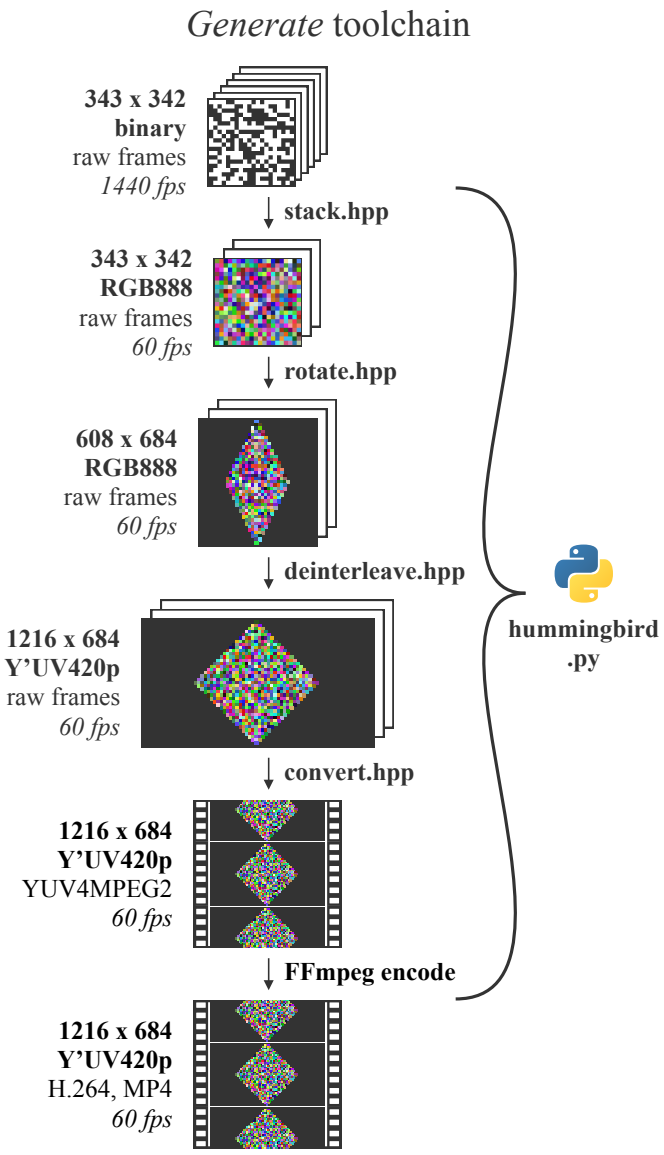


Fig. IV.9 *The generate toolchain contains tools to encode a visual stimulus in a format compatible with the play toolchain. It provides two variants: a C++ implementation that can be used from other programs, and a Python implementation compatible with Psychopy (where the latter is used to create the input binary frames). Both variants are open-source and documented on the project's Github page. This toolchain does not feature real-time features: stimuli must be encoded before running the experiment.*

similar features on most platforms, but support is not as good on the *Jetson TX1*.

We do not use the Gstreamer library to display frames, even though it is one of its features. Instead, the decoded Y*UV420p are extracted from the Gstreamer pipeline, processed using the *interleave.hpp* function (inverse of the *deinterleave.hpp* one) and sent to the HDMI output using the GLFW library [188] and the GPU. This choice is motivated by several elements. First, it is easier to extract frames from a Gstreamer pipeline than to create a custom Gstreamer filter, which would be required to implement the *interleave* step, because Gstreamer relies on the non-standard GObject system. Secondly, this gives the toolchain a lot of flexibility regarding buffering. The play program uses buffers of sixty frames, meaning that sixty frames are decoded and stored in memory before starting the video. Buffering increases the delay before starting a stimulus but reduces the risk to miss frames, which is vital for fine control over the experiments' conditions. Finally, the GLFW library features an easy way to configure the GPU's gamma ramp, responsible for color correction of the output frames. Similarly to the reason why lossy encoding must be avoided with media destined to the *DLP Lightcrafter 3000* in 1440 Hz mode, a non-identity gamma correction - default for most operating systems - will drastically change the nature of the binary patterns.

c) Running experiments

The Hibiscus module includes Hummingbird as a git submodule. It bundles C++ header-only files completing the features needed for psychophysics experiments. Two command-line programs, which use the C++ code, are provided. The first one estimates a *Livetrack* calibration using n known points (n defaults to 9) projected using the tilted *DLP Lightcrafter 3000*. The homography converting *Livetrack* coordinates to screen coordinates is calculated using the singular value decomposition method followed by a Nelder-Mead optimization using the projection errors' maximum as heuristic. The second program records user input (response box and eye tracker data) from an arbitrarily long sequence of stimuli. The *record* program includes sources synchronization, and expresses the eye tracker data in screen coordinates using the calibration. The output file, encoded using the Event Stream file format specification (see Appendix A), is ready for data interpretation. It provides a single, time-consistent sequence of multiplexed events.

The *Teensy 3.5* firmware is written using the Arduino software with the Teensyduino extension. timestamping is implemented with an interrupt for each event source (left and right buttons of the response box and the DMD trigger). Each interrupt-handling code fills a 256 bits circular FIFO emptied by the main loop. The circular FIFO implementation uses a volatile `head` index updated by the interrupt, and a non-volatile `tail` index managed by the main loop.

Hibiscus provides detailed instruction to install the software and its dependencies on out-of-the-box *Jetson TX1* and *Teensy 3.5* boards.

IV.4 Validation experiments

We perform several experiments to validate the hypothesis supporting our device, and characterize the achieved timestamp precision. The experiments use the following external devices:

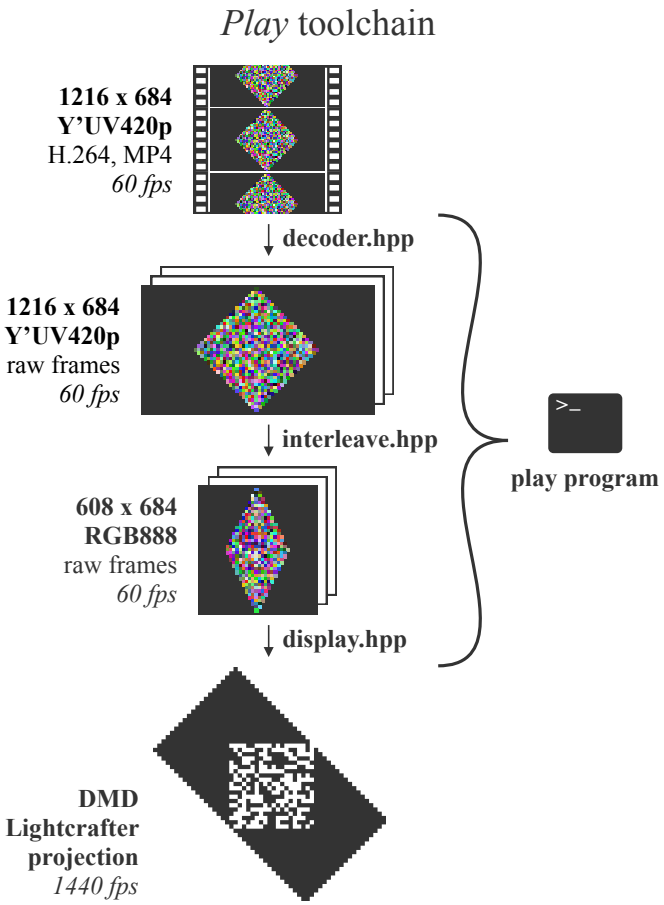


Fig. IV.10 *The play toolchain uses the output from the generate toolchain. It provides C++ header-only implementations and a program for playing stimuli using a DLP Lightcrafter 3000. If available, a hardware-accelerated H.264 decoding algorithm is used by the underlying Gstreamer library. The code is open-source and documented on the project's Github page.*

- a *Keysight InfiniiVision DSO-X 3024T* oscilloscope with a 200 MHz bandwidth and up to 5 billion samples per second
- a *Hameg HM8131-2* 15 MHz function generator, with less than 25 ns of jitter, and a 100 ppm (parts per million) frequency variation (the latter is a rough estimate accounting for the device age)
- a *Texas Instruments OPT101* photodiode and amplifier with a 14 kHz bandwidth

IV.4.1 Validating the DLP Lightcrafter 3000 driver

The *generate* and *play* toolchains presented in this work are very sensitive to lossy encoding or color corrections, because the bit planes sent to the *DLP Lightcrafter 3000* are not interpreted as colors. In order to make sure that the system displays the expected stimuli time-wise, we use the the *generate* pipeline to create a spatially uniform stimulus.

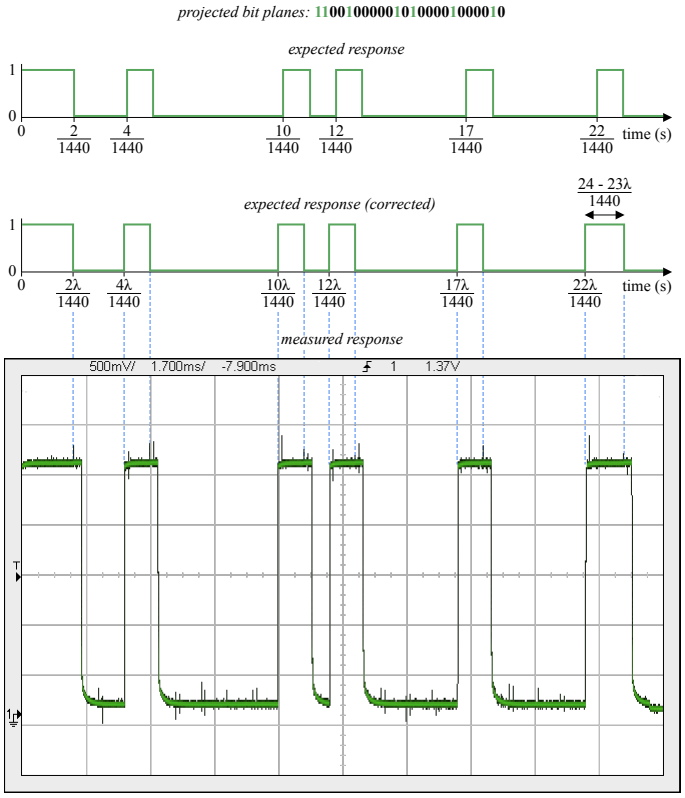


Fig. IV.11 *The DLP Lightcrafter 3000 interprets bit planes as binary patterns. In order to ensure that the correct bit planes are sent over HDMI (without color correction), we project a spatially uniform video onto a photodiode connected to an oscilloscope. Each frame's active bits form a Golomb ruler, avoiding shift invariances. The penultimate bit does not match the ruler pattern. It is set to one to illustrate a DLP Lightcrafter 3000 defect:* 1440 Hz *frames have uneven durations.*

Pixels are switched on and off in a 24 frames Golomb ruler pattern displayed at 1440 Hz, illustrated figure IV.11. The *DLP Lightcrafter 3000* projector lights up a photodiode connected to an oscilloscope. The pattern is chosen as to not be shift-invariant, reducing the risk to mistakenly obtain the correct result. Notably, bytes are not read by the *DLP Lightcrafter 3000* in the usual RGB order. A good input and output match is achieved using the correction motivated by the defect described in section IV.2.2. This defect is characterized by 23 out of 24 frames being 2.1 % too short, and 1 out 24 frames being 48 % too long. We observe the expected timings - with microsecond precision - for off-to-on transitions. However, on-to-off transitions are always 25 µs late, or 3.6 % of the average frame duration.

IV.4.2 Validating the timestamps precision

Every user input is precisely timestamped using the *Teensy 3.5* clock, either directly (left and right response box buttons, DMD trigger) or through an implicit synchronization

mechanism (frames labels, eye tracker samples). To estimate the timestamping accuracy, we send a square signal to one of the *Teensy 3.5*'s inputs, as illustrated by the left plots shown figure IV.12. The rising edges of the square signal are detected with interrupts and timestamped. We observe the evolution of two quantities with respect to the signal frequency. First, the number of missed samples is estimated by calculating the time difference between the generated timestamps. Secondly, for successfully acquired samples, we calculate the absolute time difference between the theoretical rising time of the square signal and the obtained timestamp. The *Teensy 3.5* and function generator clocks are assumed monotonic but of imprecise frequencies: they have a distinct time reference and tend to drift over time. The 100 ppm variation of the function generator clock's frequency can cause a drift up to 100 µs every second for a 1 MHz signal. Hence, timestamp errors are estimated after using a linear regression to convert the function generator times to the *Teensy 3.5* clock referential. We observe a sudden increase in the number of missed samples when reaching about 400000 interrupts per second.

Interrupts are handled sequentially by the *Teensy 3.5*. Therefore, the errors are maximized by events arriving exactly at the same time. To quantify the timestamping accuracy with such events, we connect the function generator output to three independent *Teensy 3.5* pins, each triggering a distinct interrupt. The same quantities as in the one interrupt case are measured. We observe a similar increase at 400000 interrupts per second, reached with a 130 kHz signal (since each rising edge triggers three interrupts). timestamps errors stay under 10 µs for frequencies below 100 kHz. A typical use of out box generates about 1440 interrupts per second, and rarely two interrupts at once. Thus, the timestamping is always accurate to 10 µs, and generally accurate down to 1 µs.

Similarly to function generator, the *Teensy 3.5* clock drifts over time (about 20 ppm). This defect does not impact comparative studies, but makes inter-studies comparison more difficult (as if each study had its own definition of the second, with subtle variations between studies). Long experiments where a high time accuracy matter must take this defect into account by estimating their hardware-specific drift with respect to the experiment's temperature conditions.

The synchronization with the *Livetrack* does not use interrupts, but a time measure followed by a digital write in the *Teensy 3.5*'s program main loop. This pair of instructions may be separated by several processor cycles, but the time delta remains negligible when compared to the errors induced by the *Livetrack*'s 2 ms gap between samples. Complex synchronization algorithm using multiple round-trip estimates could provide a synchronization precision below the sample gap, but cannot account for the unknown jitter of the samples. More information on the undisclosed *Livetrack* architecture is needed for meaningful improvements.

IV.4.3 Determining the parameters for frame labelling

Frame labelling cannot be handled by the *Teensy 3.5* because the DMD trigger does not provide a frame index. The implicit synchronization mechanism presented section IV.3.2 solves this issue, but relies on an unknown integer frame delay. In order to estimate this delay, we project a 1 Hz, spatially uniform binary pattern onto a photodiode. The wiring used by this experiment is illustrated figure IV.13. We compare the timestamps of the DMD trigger rising edges with the timestamps of the photodiode rising edges, in order to

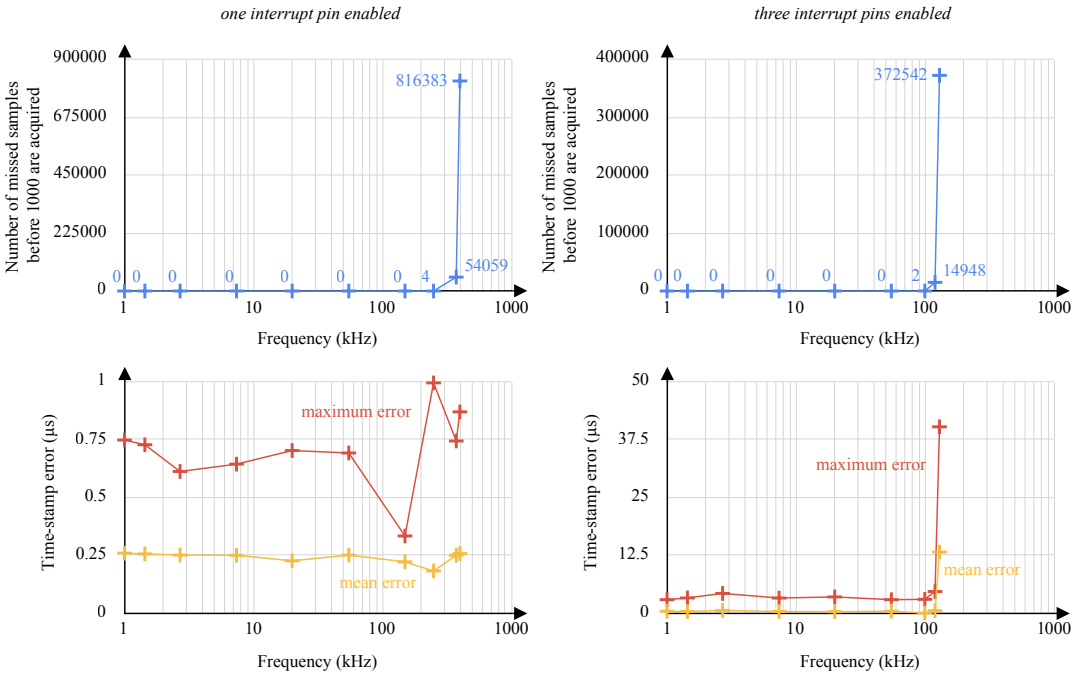


Fig. IV.12 *Precise timing relies only on the Teensy 3.5's clock, defined as reference. When an electronic interrupt is triggered by a user inputs or the DLP Lightcrafter 3000, a timestamp is generated and pushed to a FIFO, waiting for USB transmission. In order to estimate the resulting precision, we send a square signal of varying frequency to a Teensy 3.5 pin. The time delta between timestamps provides an estimation of the number of missed rising edges, shown by the top-left plot. The bottom-left plot shows the acquired samples' jitter as a function of frequency. The right column illustrates a similar experiment, where the squared signal is recorded by three independent pins. This unfavourable case comes with an increased jitter, as only a single interrupt can be handled at once. In both cases, we observe a major drop in performance around 400000 interrupts per second. However, the intended use of the box yields only about 1440 interrupts per second, and rarely more than one at a time. timestamp errors stay well below* 10 µs *under these conditions.*

find out which trigger rising edge is associated with a specific frame index. The small time difference between the timestamps allows us to undoubtedly associate each photodiode rising edge with one - and only one - DMD trigger rising edge.

The DMD rising edges are always timestamped 2 µs to 3 µs after the photodiode's, leading to the conclusion that this time difference is a systematic *DLP Lightcrafter 3000* error, rather than a jitter induced by the interrupt-based timestamping. However, this error is in the order of magnitude of the device time accuracy, and can therefore be ignored.

IV.4.4 Estimating the number of missed frames

Sending frames over HDMI to the *DLP Lightcrafter 3000* is handled by the *Jetson TX1*, which does not host a real-time operating system. The thread responsible for sending frames to the GPU at 60 Hz may be preempted by the kernel at the wrong time. If it happens, the GPU sends the same frame twice to *DLP Lightcrafter 3000*. The associated 24-frames pattern is projected twice, resulting in an unexpected motion. We call this defect *missed frame*. It can be detected thanks to the synchronization mechanism illustrated figure IV.7. In order to estimate its frequency of occurrence, we simulate the conditions of a psychophysics experiment. We use a sequence of two hundred clips generated with Psychopy's DotStim stimulus. Each clip is separated in three phases of varying duration: *fixation*, *incoherent phase* and *coherent phase*. An example is illustrated figure IV.14. In the original experiment, the subject is expected to press the response box's buttons when the coherent phase starts. To simulate the different outcomes (button pressed during any of the phases, or not pressed at all) without the need for human subjects, we draw a random number in the range $[1..100]$ with each *Livetrack* synchronization message - once every 25 ms. If the drawn number is one, we trigger a button press. This simple implementation results in a random event with the cumulative distribution function shown figure IV.14.

We run the sequence of two hundred clips five times. Since button presses stop the current clip and start the next one, each run takes a variable time, close to ten minutes. Table IV.3 shows the number of frames actually displayed in each run, and the number of detected synchronization issues. Each synchronization issue impacts 24 1440 Hz frames. We assume that any clip during which a synchronization issue occurred needs to be removed from the analysis. Some issues happen in pairs, and generally characterize a late USB packet rather than a missed frame: the associated clip was displayed as expected, and can be kept in the analysis. Nevertheless, such clips are counted as *impacted* in the results presented table IV.3. We observe a low - though non-zero - number of issues.

IV.5 Discussion

The box presented in this works bundles a projector, a computer with an embedded GPU and a real-time microprocessor. Unlike existing solutions, it does not require electronics development and does not rely on expensive components. Replicating this work requires soldering resistors, printing 3D components and cutting plastic. The tools and expertise to perform these operations, if not found in the research team, may be found in most fab labs. One can also avoid assembly altogether by powering the components individually.

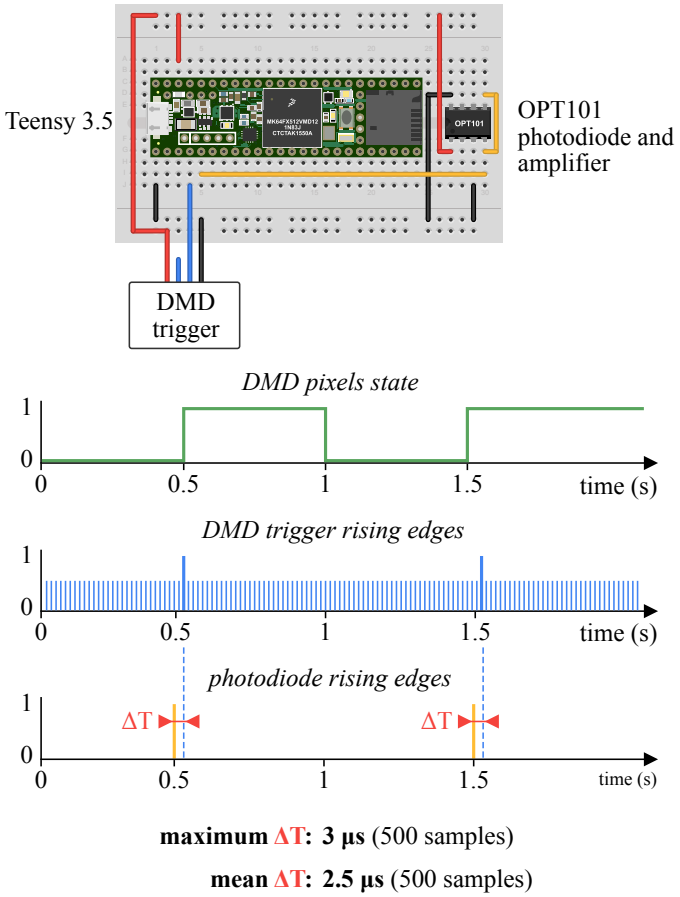


Fig. IV.13 *The DMD trigger tells the Teensy 3.5 when frames are displayed, but not which frame is being displayed. The Jetson TX1 knows when the function* `swap_buffers` *returns in the Teensy 3.5's clock referential, thanks to the mechanism illustrated figure IV.7. However, the frame is actually displayed by the DLP Lightcrfater 3000 after a specific integer number of frames. In order to estimate this delay, we project a spatially uniform* 1 Hz *onto a photodiode. The Teensy 3.5's interrupts are used to compare the arrival times of the DMD trigger and photodiode rising edges. The photodiode's rising edges match the DMD trigger's with a* 3 microsecond *precision. The time delta is almost constant, and the photodiode's interrupt is always trigger before the DMD trigger's. Therefore, we can label specific DMD trigger rising edges and match them with a frame index, since we know which frame in the stimulus corresponds to a light increase. The measured frame delay is two.*

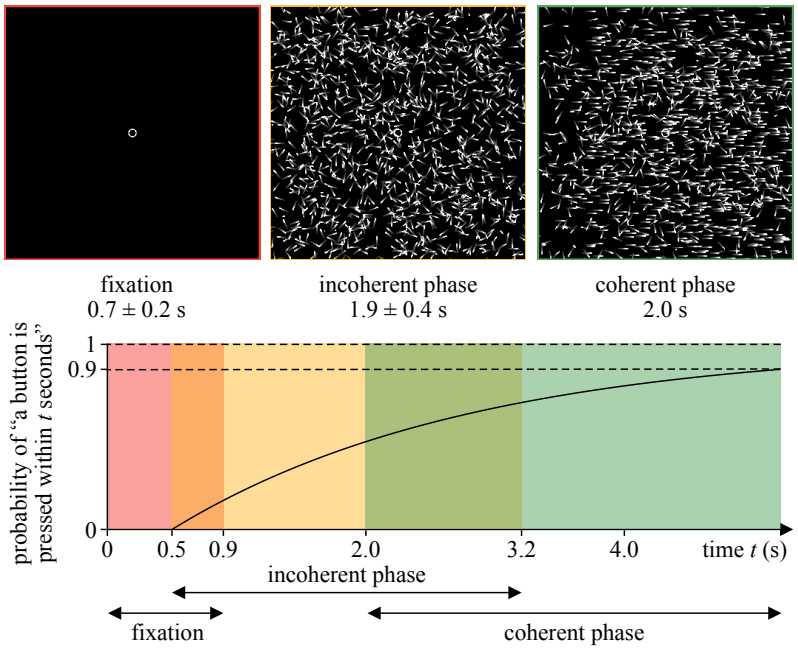


Fig. IV.14 *In order to evaluate the box behaviour under realistic conditions, we simulate an actual psychophysics experiment. The generated stimulus, which consists in moving dots, has three phases shown in the first row. In order to represent motion, several frames are superimposed with a time decay for each phase. Responses of human subjects are roughly approximated with a random draw at fixed time intervals, resulting in the cumulative distribution shown in the second row.*

trial	1440 Hz frames	unsynchronized 60 Hz frames	1440 Hz frames impacted	clips impacted
1	686016	5	0.02 %	1.5 %
2	701904	0	0 %	0 %
3	723744	0	0 %	0 %
4	727728	1	0.003 %	0.5 %
5	719928	2	0.007 %	1.0 %
mean	**711864**	**1.6**	**0.006 %**	**0.6 %**

Table IV.3: The stimulus illustrated figure IV.14 is used to generate two hundred clips. The clips are associated in a sequence repeated over five trials, with random button pushes simulating a human subject. We calculate the number of unsynchronized frames, and, assuming that a missed frame invalidates the stimulus in which it occurred, the number of clips lost due to the system limits. For the intended use (many repetitions of short clips), the average loss rate is acceptable.

However, this would result in a less stable, bulkier experimental setup. Such setups, or experiments which do not require precisely-timed recordings, still need custom parts to tilt the *DLP Lightcrafter 3000* at a 45 degrees angle and precisely move its focus. Appendix B presents a 3D printed kit fulfilling this need. The parts are compatible with metric optical tables, and feature a wheel to control the projector's focal length.

The introduced code toolchains assume an offline stimulus generation. A real-time, configurable generator could be beneficial in several situations. The development of novel stimuli would be easier, allowing a shorter iterative design cycle. Moreover, the possibility of experiments where feedback is used to adjust the stimulus arises. Even though real-time stimulus generation is possible with the considered hardware, usual stimulus generators are not optimized for the frames expected by the *DLP Lightcrafter 3000* in 1440 Hz mode, and fall short of running in real-time. Consequently, the code generating the stimulus must be written from scratch in a "low-level" (C++ rather than Python) programming language, which is both time-consuming and of uncertain outcome: if the stimulus requires to much computation, off-line pre-processing may still be needed.

The projector used in this work was chosen as the least expensive solution to display 1440 Hz frames. Less expensive DMD-based projectors from Texas Instruments[7] do not feature the required HDMI to binary pattern operating mode. More expensive projectors[8] provide the needed features, and avoid the need for the 45 degrees tilt workaround since they have a usual, non-diamond pixel pattern. This work's other considerations, in particular encoding and synchronization, are still relevant.

Displays' maximum frame rate used to be limited by the pixels response time. Novel technologies, such as DMDs and LEDs, have response times so small that the limiting factor for high-speed screens becomes the computer-to-display communication bandwidth, and the computer's capacity to generate frames. The *DLP Lightcrafter 3000* circumvents this limit by interpreting color depth as binary patterns, giving up color for speed. A similar technique could be applied to LED screens. With the long-term goal of building a display indistinguishable from a real scene by the human eye, the maximum frame-rate we are sensitive to has major consequences in technology design. The faster the frame-rate, the fewer pixels change with each frame, and the more full frames of information are redundant. Alternatives to the usual, well-established pipeline (GPU connected to a display with a fixed frame-rate) become competitive. They range from the variable frame-rate used by Nvidia's Gsync technology, to the "frames interpreted as temporal patterns" technique implemented by the *DLP Lightcrafter 3000*, to a hypothetical fully asynchronous event-based display.

Other research fields than psychophysics can benefit from neuromorphic engineering methods and devices. chapter III presented a three-chip event-based camera and a color segmentation algorithm. The next chapter discusses the application of this sensor to *Brainbow* [21], a bio-engineering method generating complex fluorescent neuron samples that represent a challenge for conventional segmentation algorithms.

[7] http://www.ti.com/lit/ug/dlpu049c/dlpu049c.pdf
[8] http://www.ti.com/lit/ug/dlpu011f/dlpu011f.pdf

Chapter V

A Markov chain model for Brainbow

V.1 Introduction

Brainbow [21] uses bio-engineering to insert specific transgenes in mice DNA. The modified mice are bred to produce genetically modified specimens with a replica of the transgene of interest in every cell. At an arbitrary time during the bred specimens' development, a chain of chemical processes called Cre-*lox* recombinations is externally triggered. These chemical reactions mutate the inserted transgenes, converting them to new DNA arrangements randomly drawn from a finite set, which depends on the arrangement of lox sites. The arrangements obtained at the end of the chain of reactions are called *outcomes*. In the case of Brainbow, the transgenes are chosen so that each outcome expresses a different gene producing fluorescent proteins sensitive to a specific wavelength. Since each cell is randomly assigned an outcome, it will be randomly sensitive to one among several wavelength, resulting in multicolored brain samples such has the one shown figure I.8.

By artificially creating DNA differentiation from a genetic code identical in every cell, Brainbow helps with neural images segmentation and DNA barcoding [189]. Moreover, since the process can be triggered during development, it is possible to randomly assign outcomes to individual progenitors instead of neurons. Since progenitors pass on their DNA to the neurons they produce, cells with a common progenitor will express the same fluorescent proteins, and have the same color in the nervous system. This property can be used to advance our understanding of development, and explore the relation between neural development (how the brain grows) and the connectome (the map of neural connections).

Automated segmentation of Brainbow images remains a challenge for conventional computer vision: the solutions provided by pre-deep learning methods are not very robust, whereas deep learning approaches require large amounts of labelled data. The new approach to segmentation made possible by color event-based sensors is promising to solve the problem, all the more so as the amount of fluorescent proteins varies from neuron to neuron, thus a high dynamic range sensor improves the quality of captured data. However, while Brainbow samples - like other biological samples - deteriorate when lit with strong light sources, the ATIS used in our color camera has little sensitivity under low-light conditions. The measurements we performed on Brainbow samples with an ATIS did not result in a satisfactory outcome. Recent works tackled low-light sensitivity

in event-based cameras [172, 190], however these methods have yet to be extended to event-based cameras with absolute luminance measurement capabilities.

The process associated with a large chain of Cre-*lox* recombinations is complex. The hand-crafted transgenes leveraged by Brainbow have a small number of *lox* sites, by lack of a mathematical model to predict the outcomes of more complex transgenes. Yet, the latter would yield more balanced color distributions and enable the expression of more colors. Assuming sensitive enough event-based cameras, transgenes' complexity may become the limiting factor in improving Brainbow. To prepare the ground for future event-based cameras, this chapter presents a new mathematical model to predict the outcomes distribution of a Cre-*lox* transgene - that is, the different states of the DNA sequence after a specific number of Cre-*lox* recombinations, and their probability of occurrence. We use this model to predict the outcomes of every possible transgene containing up to nine *lox* sites and four *lox* types, yielding a database that can be searched for transgenes with specific properties. Both the simulation code and the database will be made freely available online.

A lox site is a non-symmetric 34 base pairs DNA segment. Two lox sites in presence of Cre recombinase form a chemical complex which reroutes DNA. If both sites have the same orientation, the DNA portion between the sites is irreversibly excised. Otherwise, the DNA portion between the sites is inverted. A new reaction between the two same sites would revert the DNA segment to its initial orientation. The inversion and excision mechanisms are illustrated figure V.1. When more than two lox sites appear in a DNA segment, the presence of Cre recombinase triggers a sequence of recombinations. Several lox sites types that cannot recombine have been identified and synthesized. Different sequences of recombinations applied to the same transgene can yield distinct DNA segments. For example, the *Brainbow 2.1 transgene*, illustrated figure V.2, has four outcomes. An outcome is determined by the recombinations order, which changes randomly between cells.

Former studies on many-lox transgenes rely on discrete time Monte-Carlo simulations to predict the considered transgene's outcomes and probabilities [189, 191, 192]. This approach has two shortcomings: it does not account for the variation of the overall recombination rate as a function of the number of lox pairs, and it only yields an approximated estimation of the stationary distribution (the outcomes probabilities after an infinite time). Y. Wei et al. [193] proposed a formal model, and used it to identify configurations propitious for cellular barcoding. They mathematically demonstrate their model for *shufflons* - transgenes with only inversions - under reasonable assumptions. In order to extend the model to excisions, they assume that the latter's dynamics are slow compared to inversions. This is a strong assumption, provided that excisions and inversions are consequences of the same chemical reaction. In particular, it can lead to important errors in the estimation of the probabilities of occurrence for transgenes with both excisions and inversions. However, such transgenes are extensively used by Brainbow [194], and in *Polylox* transgenes [195, 191]. Moreover, this model does not apply to transgenes containing multiple incompatible lox sites types.

The model presented is this work is based on the same assumptions as Y. Wei et al., but accounts for excisions without assuming slow dynamics compared to inversions, and applies to transgenes with several lox sites types. It can take into account a minimum recombination distance, in order to model the recombination probability variation with

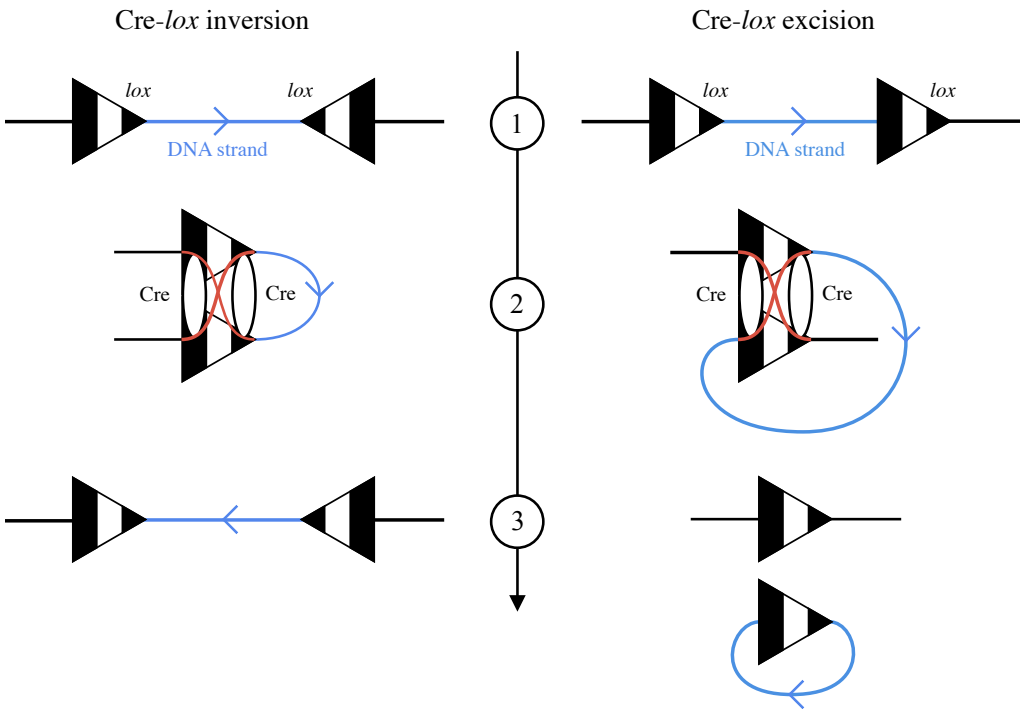


Fig. V.1 *The initial configurations for Cre-lox inversion (left) and Cre-lox excision (right) are shown (1). The DNA strand between the two sites may contain lox sites that will be impacted by the transformation. Cre proteins create a complex between the lox sites, re-routing the original DNA. The chemical reaction is identical for both mechanisms (excision and inversion). Only the sites' relative orientation impacts the result (2). After the transformation, the DNA strand between the lox sites is inverted if they had distinct orientations, and forms a detached loop if they had the same orientation (3). The inversion mechanism is reversible, whereas the excision is not. This work assumes a first order chemical reaction for Cre-lox recombinations, with a constant rate* λ.

respect to the distance between sites. In order to identify optimal transgenes for various applications, we provide a database containing every possible transgene with up to nine lox sites and four lox sites types, and their outcomes' distribution calculated with the model. This database assumes that there is no minimal distance for recombination. We provide a second database, which assumes a minimum distance of 82 base pairs between sites to allow recombination. It contains every possible transgene with up to seven lox sites, three lox types and each relevant combination of lengths for the DNA strands between lox sites. Both databases can be searched to find the transgenes best suited to a given application by calculating a score function on each of their entries.

V.2 Methodology

Cre-*lox* recombinations applications require transgenes with specific properties. As an example, Brainbow is likely to yield better results with transgenes whose outcomes are

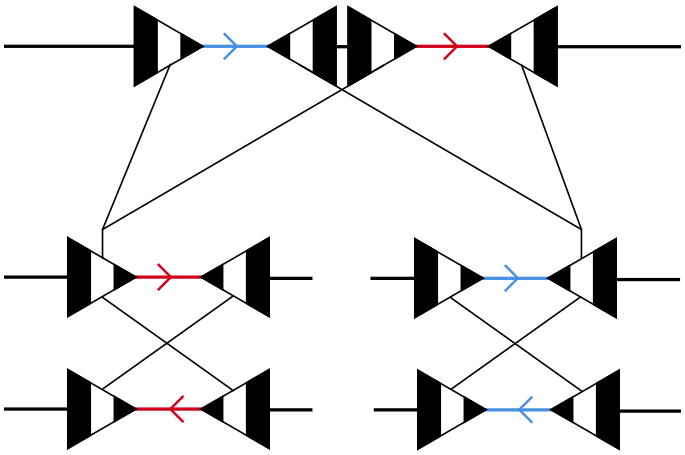


Fig. V.2 *The Brainbow 2.1 transgene (top line) contains four lox sites and two genes expressing different fluorescent proteins (blue and red). Depending on which excision happens first, a neuron will be left with either the blue or the red gene, and express only one type of fluorescent proteins (bottom lines).*

irreversible and equiprobable. In order to identify well suited transgenes, we use a two-steps approach. First, we provide a method to determine a DNA strand's outcomes and calculate their probability of occurrence, or weights. Then, we apply this method to every *lox* configuration that can be generated from a given set of *lox* sites, and find among them the most relevant to the task of interest.

V.2.1 Markov process

We assume that any pair of compatible *lox* sites has a constant recombination rate λ, meaning that its probability to recombine over a small time window Δt is equal to $\lambda \cdot \Delta t$. This hypothesis implies that transgenes containing *lox* sites have no memory: the recombination probabilities of a transgene's *lox* sites pairs do not depend on the recombinations that led to said transgene. The minimum recombination distance is accounted for section V.3.2. Under this assumption, the different DNA strands derived from an initial one can be represented as the n states of a continuous-time Markov process. The latter's infinitesimal generator $\boldsymbol{A}$ is the $n \times n$ matrix with elements $A_{ij}, (i,j) \in [\![1,n]\!]^2$ given by:

$$A_{ij} = \begin{cases} 1 - \sum\limits_{k \neq i} A_{ik} & \text{if } i = j, \\ \lambda r_{ij} & \text{otherwise.} \end{cases} \tag{V.1}$$

r_{ij} is the number of distinct Cre-*lox* recombinations (excisions or inversions) turning the strand i into the strand j. r_{ij} is often either 0 or 1. As an example, the Brainbow-2.1

Index	Transgene
1	▶A→◀▶B→◀
2	▶A→◀▶B←◀
3	▶A←◀▶B→◀
4	▶A←◀▶B←◀
5	▶B→◀▶A→◀
6	▶B→◀▶A←◀
7	▶B←◀▶A→◀
8	▶B←◀▶A←◀
9	▶A→◀
10	▶A←◀
11	▶B→◀
12	▶B←◀

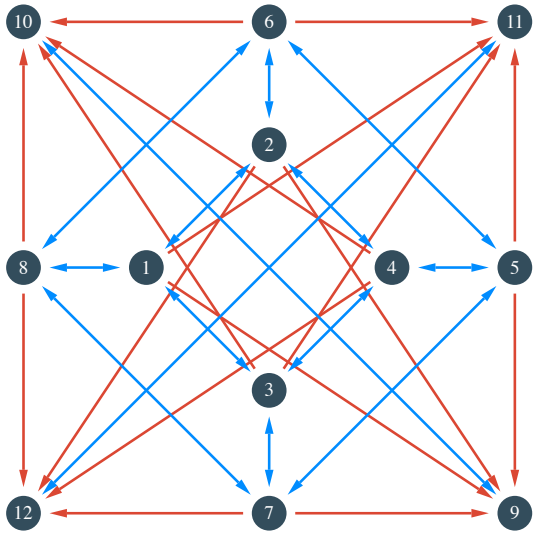


Fig. V.3 *The table on the left lists the Brainbow-2.1 transgene's recombined forms. The initial state has index 1. The graph on the right illustrates the Markov process derived from the Cre-lox recombinations. Bi-directional blue arrows represent inversions, whereas red arrows represent excisions.*

transgene has the infinitesimal generator:

$$\begin{pmatrix}
\alpha & \lambda & \lambda & 0 & 0 & 0 & 0 & \lambda & \lambda & 0 & \lambda & 0 \\
\lambda & \alpha & 0 & \lambda & 0 & \lambda & 0 & 0 & \lambda & 0 & 0 & \lambda \\
\lambda & 0 & \alpha & \lambda & 0 & 0 & \lambda & 0 & 0 & \lambda & \lambda & 0 \\
0 & \lambda & \lambda & \alpha & \lambda & 0 & 0 & 0 & 0 & \lambda & 0 & \lambda \\
0 & 0 & 0 & \lambda & \alpha & \lambda & \lambda & 0 & \lambda & 0 & \lambda & 0 \\
0 & \lambda & 0 & 0 & \lambda & \alpha & 0 & \lambda & 0 & \lambda & \lambda & 0 \\
0 & 0 & \lambda & 0 & \lambda & 0 & \alpha & \lambda & \lambda & 0 & 0 & \lambda \\
\lambda & 0 & 0 & 0 & 0 & \lambda & \lambda & \alpha & 0 & \lambda & 0 & \lambda \\
 & & & & & & & & \beta & \lambda & & \\
 & & & & & & & & \lambda & \beta & \mathbf{0} & \\
 & & & \mathbf{0} & & & & & & & & \\
 & & & & & & & & & & \beta & \lambda \\
 & & & & & & & & \mathbf{0} & & & \\
 & & & & & & & & & & \lambda & \beta
\end{pmatrix}$$

$\alpha = -5\lambda$ and $\beta = -\lambda$ are used here to improve readability.

The Markov process' states distribution is defined as the probability of each strand, and changes over time. The initial distribution is such that the initial transgene has probability 1, and all the derived strands have probability 0.

V.2.2 Absorbing sets

Stopping Cre-lox reactions when reaching an arbitrary progression is not easy: the reaction may not begin at the same moment in every cell, and late recombinations can be triggered by leftover recombinase. A simple solution consists in assuming an excess

recombinase, and wait for a long time. These conditions are described by the limiting distribution of the Markov process built from a Cre-lox transgene. The latter is defined as the states' probability distribution after an infinite amount of time.

Reibman et al. [196] introduced a numerical method to calculate the Markov process's states' distribution after an arbitrary, but finite, amount of time. Using this method to calculate the limiting distribution requires an infinite amount of matrix multiplications. Instead, in order to determine the limiting distribution associated with a given transgene, we leverage *absorbing sets*. An absorbing set S is a set of states with the following properties:

- for any two states $s_1, s_2 \in S^2$, there is a sequence of inversions turning s_1 into s_2.
- for any state $s \in S$, s cannot be excised.

As an example, the Brainbow-2.1 transgene has two absorbing sets: $\{9, 10\}$ and $\{11, 12\}$. States that are not part of an absorbing set are called *transient*.

Since all the states of an absorbing set communicate (directly or not), the Markov process formed by its strands is irreducible, therefore ergodic (as a continuous-time Markov process). As such, its limiting distribution is equal to its stationary distribution, and is unique. Moreover, since every transition within the set is an inversion, its infinitesimal generator is symmetrical. The uniform distribution is a stationary distribution for a symmetrical generator [197]. Therefore, it is the unique limiting distribution of any absorbing set.

In other words, there is an equal probability to observe each state of an absorbing set after a large number of recombinations, regardless the initial distribution.

V.2.3 Reduction to an absorbing process

The states of any Markov process built from a Cre-*lox* transgene can be re-ordered so that absorbing sets' states have consecutive indices. The process's infinitesimal generator $\boldsymbol{A}$ can then be written as:

$$\boldsymbol{A} = \left(\begin{array}{c|ccc} \boldsymbol{T} & \multicolumn{3}{c}{\boldsymbol{R}} \\ \hline & \boldsymbol{A_1} & & \\ & & \boldsymbol{A_2} & \boldsymbol{0} \\ \boldsymbol{0} & & & \\ & \boldsymbol{0} & & \ddots \\ & & & \boldsymbol{A_k} \end{array}\right) \tag{V.2}$$

$\boldsymbol{T}$ is a $t \times t$ matrix, where t is the process's number of transient states. k is the number of absorbing sets, and the matrix $\boldsymbol{A}_i, i \in [\![1, k]\!]$ is the infinitesimal generator associated with the absorbing set i. $\boldsymbol{R}$ is a $t \times (n - t)$ matrix, where n is the total number of states.

Such a Markov process's limiting distribution exists and is a stationary distribution [198], regardless the initial states' distribution. Let $\boldsymbol{\pi}$ be such a distribution. It can be expressed as:

$$\boldsymbol{\pi} = (\boldsymbol{0}_t \; p_1\boldsymbol{\pi_1} \; \ldots \; p_k\boldsymbol{\pi}_k) \tag{V.3}$$

$\boldsymbol{0_t}$ is the null vector with t components. $\boldsymbol{\pi_i}, i \in [\![1, k]\!]$ is the limiting distribution for the Markov processes associated with the infinitesimal generators $\boldsymbol{A_i}$. As shown previously,

they are uniform distributions. $p_i, i \in [\![1, k]\!]$, are scalar coefficients which depend on the initial states' distribution, and sum to one. Therefore, the limiting probability p_s for a state s in the absorbing set i is given by:

$$p_s = \frac{p_i}{|\boldsymbol{A_i}|} \tag{V.4}$$

$|\boldsymbol{A}_i|$ is the number of states in the absorbing set i.

In order to calculate the p_i coefficients, we build a new Markov process, called *reduced*, with states $S^\star$ according to the following rules:

- For any state $s \in S$, $s \in S^\star$ if and only if s is transient.
- Each absorbing set of S is associated with a single state of $S^\star$. This state has no outgoing transitions, and is said to be *absorbing*.
- The transition rate from a transient state to another is identical in S and $S^\star$.
- The transition rate from a transient state s_t to an absorbing state s_a in $S^\star$ is equal to the sum of the transitions rates from s_t to each state of the absorbing set associated with s_a in S.

These rules can be understood as the merging of each absorbing set to a single state. As an example, the reduced Markov process for the Brainbow-2.1 transgene has states $S^\star = \{1, 2, 3, 4, 5, 6, 7, 8, 1^\star, 2^\star\}$, illustrated figure V.4. The states $1^\star$ and $2^\star$ are associated with the absorbing sets $\{9, 10\}$ and $\{11, 12\}$ of the original Markov process, respectively. The reduced process has the infinitesimal generator:

$$\begin{pmatrix}
\alpha & \lambda & \lambda & 0 & 0 & 0 & 0 & \lambda & \lambda & \lambda \\
\lambda & \alpha & 0 & \lambda & 0 & \lambda & 0 & 0 & \lambda & \lambda \\
\lambda & 0 & \alpha & \lambda & 0 & 0 & \lambda & 0 & \lambda & \lambda \\
0 & \lambda & \lambda & \alpha & \lambda & 0 & 0 & 0 & \lambda & \lambda \\
0 & 0 & 0 & \lambda & \alpha & \lambda & \lambda & 0 & \lambda & \lambda \\
0 & \lambda & 0 & 0 & \lambda & \alpha & 0 & \lambda & \lambda & \lambda \\
0 & 0 & \lambda & 0 & \lambda & 0 & \alpha & \lambda & \lambda & \lambda \\
\lambda & 0 & 0 & 0 & 0 & \lambda & \lambda & \alpha & \lambda & \lambda \\
 & & & & & \mathbf{0} & & & &
\end{pmatrix}$$

$\alpha = -5\lambda$ is used here to improve clarity.

By construction, the reduced process is absorbing. Its infinitesimal generator $\boldsymbol{A}^\star$ is:

$$\boldsymbol{A}^\star = \begin{pmatrix} \boldsymbol{T} & \boldsymbol{R}^\star \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{V.5}$$

$\boldsymbol{R}^\star$ can be computed from R. The process's limiting distribution $\boldsymbol{\pi}^\star$ can be expressed as:

$$\boldsymbol{\pi}^\star = (\mathbf{0_t}\ p_1\ \ldots\ p_k) \tag{V.6}$$

The p_i coefficients are the same as the coefficients appearing in the original Markov process' limiting distribution. Since the reduced Markov process $S^\star$ is absorbing, it has

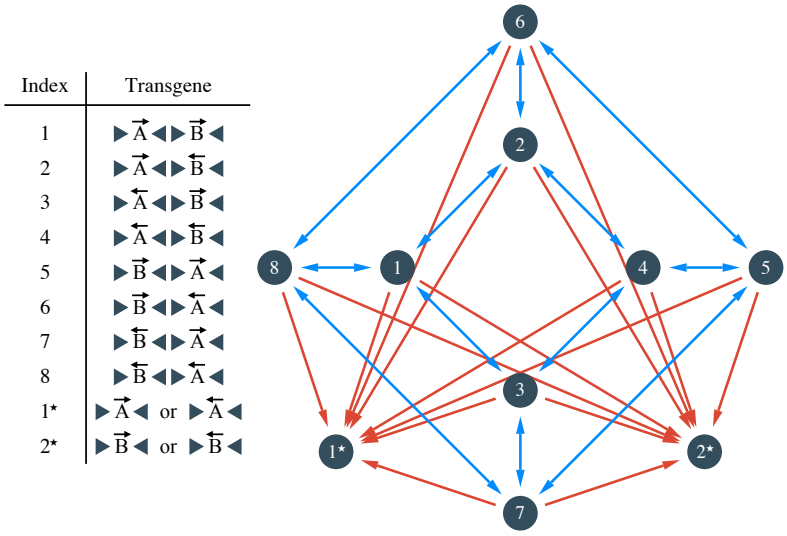


Fig. V.4 *The table on the left lists the reduced Brainbow-2.1 transgene's states. The states* $1^\star$ *and* $2^\star$ *correspond to the absorbing sets of the original process. The graph on the right shows the resulting Markov process, which is absorbing. Bi-directional blue arrows represent inversions, whereas red arrows represent excisions.*

an identical stationary distribution as its associated embedded Markov chain. The latter is defined as the discrete-time Markov chain with transition matrix $\boldsymbol{P}_e^\star$ given by [199]:

$$\boldsymbol{P}_e^\star = \begin{pmatrix} \boldsymbol{T}_e & \boldsymbol{R}_e^\star \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \tag{V.7}$$

$\boldsymbol{T}_e = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{T}$ and $\boldsymbol{R}_e^\star = -\boldsymbol{D}^{-1}\boldsymbol{R}^\star$. $\boldsymbol{D}$ is a $t \times t$ matrix with elements $D_{ij}, (i,j) \in [\![1,n]\!]^2$ given by:

$$D_{ij} = \begin{cases} T_{ij} & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{V.8}$$

Interestingly, $\boldsymbol{T}_e$ and $\boldsymbol{R}_e^\star$ do not depend on λ. As an example, the embedded Markov chain for the Brainbow-2.1 transgene has the transition matrix:

$$\begin{pmatrix} 0 & 1/5 & 1/5 & 0 & 0 & 0 & 0 & 1/5 & 1/5 & 1/5 \\ 1/5 & 0 & 0 & 1/5 & 0 & 1/5 & 0 & 0 & 1/5 & 1/5 \\ 1/5 & 0 & 0 & 1/5 & 0 & 0 & 1/5 & 0 & 1/5 & 1/5 \\ 0 & 1/5 & 1/5 & 0 & 1/5 & 0 & 0 & 0 & 1/5 & 1/5 \\ 0 & 0 & 0 & 1/5 & 0 & 1/5 & 1/5 & 0 & 1/5 & 1/5 \\ 0 & 1/5 & 0 & 0 & 1/5 & 0 & 0 & 1/5 & 1/5 & 1/5 \\ 0 & 0 & 1/5 & 0 & 1/5 & 0 & 0 & 1/5 & 1/5 & 1/5 \\ 1/5 & 0 & 0 & 0 & 0 & 1/5 & 1/5 & 0 & 1/5 & 1/5 \\ & & & & & & & & 1 & 0 \\ & & & & \boldsymbol{0} & & & & & \\ & & & & & & & & 0 & 1 \end{pmatrix}$$

The absorbing states distribution $(p_0 \ \dots \ p_k)$ can be calculated from the initial transient states distribution $\boldsymbol{\pi}_t$, which is a t-components vector, with the relationship [200]:

$$(p_0 \ \dots \ p_k) = \boldsymbol{\pi}_t \left(\boldsymbol{I} - \boldsymbol{T_e}\right)^{-1} \boldsymbol{R_e^\star} \tag{V.9}$$

V.2.4 Algorithm

From the foregoing, we devise the following strategy to find the limiting distribution of a Cre-*lox* transgene:

1. Build the Markov process associated with the transgene. A detailed algorithm to build this process is given appendix C.1. The reaction rate λ can be given an arbitrary, non-null value.
2. Determine the process's absorbing sets. If there is a single absorbing set, the limiting distribution is the uniform distribution over this set, and the strategy ends here.
3. Build the reduced Markov process by merging the absorbing sets. A detailed algorithm to build the reduced process is given appendix C.2.
4. Calculate the embedded chain's transition matrix.
5. Calculate the absorbing states distribution, with an initial transient states distribution equal to one for the transgene of interest, and null for the others.
6. The limiting probability of each transgene in an absorbing set is given by the corresponding state's limiting probability divided by the number of transgenes in the set.

As an example, the Brainbow-2.1 transgene absorbing states' distribution is $(0.5, 0.5)$. Therefore, the transgene has an equal probability to turn into any of the states $9, 10, 11$ and 12 after a large number of recombinations.

V.3 Database

V.3.1 Every possible transgene

We aim to find optimal Cre-*lox* transgenes for several applications, such as barcoding or stochastic gene activation. These application require either a large number of outcomes, or outcomes with specific properties: an equiprobable distribution, being irreversible, a different left-most element for each outcome... For a given application, it is easy to assign a score to a transgene once its outcomes' distribution is known. The optimal transgene has the largest score. Usual optimization techniques require a score function which is regular in the transgene space: two similar transgenes (with respect to a given metric) should have similar scores. Unfortunately, similar transgenes with respect to usual metrics, such as the Levenshtein distance, can have very different scores. As an example, the Brainbow-2.1 transgene, well suited for Brainbow applications, would be given a good score. However, changing the orientation of its latest *lox* site yields a transgene with a single outcome with a single *lox* site, and thus a bad score.

Since finding a distance function with the expected regularity properties is not easy, we cannot apply efficient optimization techniques such as gradient descent, the Nelder-Mead method or the Broyden-Fletcher-Goldfarb-Shanno algorithm. Instead, we generate every possible transgene with less than a given number of *lox* sites, and calculate the score function for each of them. This solution requires much more computational power, however it works in spite of the bad properties of the transgenes' space. In order to reduce computation costs, we first generate a common database containing every possible transgene and their outcomes' distribution, determined with the method from section V.2. Finding an optimal transgene for a problem can be reduced to the calculation of each database entry's score, which generally requires significantly less computation than the database generation.

We apply the following steps to generate every possible transgene:

1. For each number of *lox* sites, we determine the possible repartitions of each incompatible *lox* types. A repartition is interesting if there are at least two *lox* sites of each type, so that recombinations may happen. Since the *lox* types are assumed equivalent, the number of elements always decreases with the type index. An algorithm to generate the repartitions associated with an arbitrary number of *lox* sites is described appendix C.3.

2. For each repartition, we determine each unique sites permutation. This is achieved by looping over the permutations in lexicographic order, since computing the next lexicographic permutation of a character chain is implemented by many programming libraries.

3. For each permutation, we compute every possible *lox* sites orientations. A transgene with n *lox* sites has 2^n possible orientations. They are given by the binary representations of each integer i in the range $[\![0, 2^n - 1]\!]$. We assign a left direction to the site with index k if the bit of i with index k is zero, and a right direction otherwise. We filter out transgenes with a first and last sites of identical type and orientation, as they yield a single outcome made of a single *lox* site.

4. The second and third steps generate redundant transgenes when two types are assigned an identical number of sites. We add a normalization step to filter out such transgenes. The normalization algorithm is described appendix C.4. A given transgene must be normalized with this algorithm to be searched in the database.

The sites repartition, number of generated transgenes and number of normalized transgenes in the database are illustrated V.5.

V.3.2 Minimum recombination distance

Up to this point, we considered that any pair of compatible *lox* sites was able to recombine. However, since the DNA molecule is not infinitely flexible, a minimum distance between sites is required for a recombination to take place [201]. Moreover, the recombination probability slowly decreases with distance [202]. As a first approximation, we consider that a pair of *lox* sites has a constant recombination rate λ if the distance between the sites is larger than 82 base-pairs, and zero otherwise. The results presented section V.2 still hold,

lox sites	*lox* types	transgenes	transgenes after normalization
2	1	4	1
3	1	8	1
4	1	16	3
	2	96	8
5	1	32	4
	2	320	33
6	1	64	10
	2	2240	170
	3	5760	66
7	1	128	16
	2	7168	694
	3	26880	752
8	1	256	36
	2	39424	2956
	3	250880	6916
	4	645120	844
9	1	512	64
	2	125952	11932
	3	1892352	52760
	4	3870720	18552
total		**6867932**	**95818**

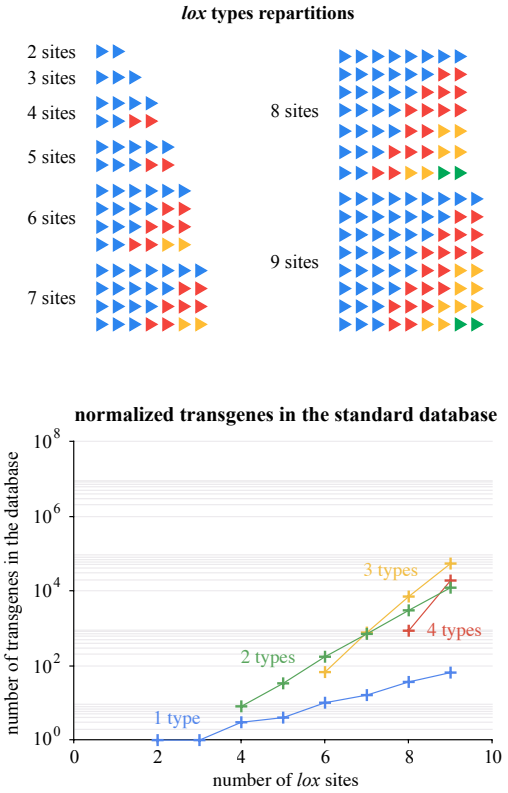


Fig. V.5 *We generate each interesting repartition for different number of lox sites (left). A repartition is interesting if there are at least two sites of each assigned type (top right). A recursive algorithm to generate the repartitions for an arbitrary number of sites is given appendix C.3. The number of possible transgenes grows exponentially with the number of sites (bottom right).*

but the graph associated with the Markov process has fewer edges: some recombinations are forbidden, because the distance between the sites involved is too small. Nevertheless, the outcomes' distribution can be calculated with the same algorithm.

The lengths of the DNA strands separating the *lox* sites impact the nature of the outcomes and their distribution. Therefore, an exhaustive database of transgenes and their outcomes which takes into account a minimum recombination distance needs to contain an entry for each possible lengths assignment for each transgene. Under our assumption that the recombination rate is constant when the distance between sites is larger than the minimum distance, there is no need to consider lengths over the minimum distance for DNA strands between sites. The number of distinct lengths assignments grows quickly with the number of *lox* sites: with a 82 base-pairs minimum distance, a 6-sites transgene has 5 strands between sites, thus $82^5 \approx 4 \times 10^9$ lengths assignments.

In order to reduce the combinatory complexity while considering every distinct lengths assignment, we define a set of inequalities to represent recombinations tipping points. As an example, let $l_1\ s_1\ l_2\ s_2\ l_3\ s_3\ l_4\ s_4\ l_5$ be a generic 6-sites transgene. $l_i, i \in [\![1,5]\!]$ are *lox* sites with arbitrary types and orientations, and $s_j, j \in [\![1,4]\!]$ are DNA strands with arbitrary lengths. We call d_j the length of s_j. By evaluating the distances between

sites pairs, we obtain a series of constrains for recombination. l_1 and l_2 can recombine if $d_1 \geq 82$. l_1 and l_3 can recombine if $d_1 + 34 + d_2 \geq 82$. l_1 and l_4 can recombine if $d_1 + 34 + d_2 + 34 + d_3 \geq 82$. l_1 and l_5 can always recombine, since the cumulated length of three *lox* sites is larger than 82 base-pairs.

Applying the same reasoning to each site, pair and triplet of non-*lox* element of the generic 6-sites transgene yields the set of inequalities shown figure V.6. If two lengths assignments respect the same constrains, then the model will predict the same outcomes' distribution for both assignments. Therefore, it is enough to consider each constrains assignment to study each lengths assignment. For a 6-sites transgene, there are 14 inequalities, thus $2^{14} = 16384$ constrains assignments, far less than the number of lengths assignments. Moreover, some constrains assignments yield incompatible inequalities. As an example, $d_1 \geq 82$ is not compatible with $d_1 + d_2 < 48$. Therefore, we determine which constrains assignments are valid for each number of *lox* sites before building a second database. Integer solutions can be found by expressing the inequalities system as an Integer Linear Programming problem, which can be solved using one of the many methods available [203]. We observe that only a fraction of the possible systems have solutions. The number of possible and solvable systems for each number of *lox* sites are listed figure V.6.

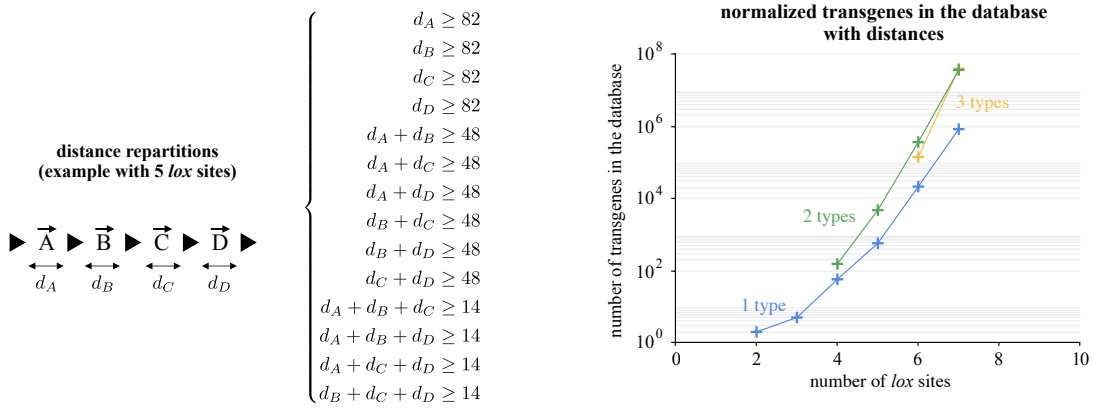
The second database generation follows the same steps as the first one. An extra step is added after transgene normalization: for each normalized transgene, DNA strands between *lox* sites are assigned the lengths associated with each solvable system.

V.4 Results

V.4.1 Experimental validation

Our model's demonstration depends on hypothesis which are known to be approximations. In order to estimate the impact of the approximations on the model's predictions, we proceed with experimental validations. For Brainbow transgenes, the model predicts the same outcomes and distributions as existing methods: all the outcomes of these transgenes are equiprobable. This prediction is compatible with experimental observations [204].

Pei et al. [191] performed experimental measurements on Polylox transgenes, but observe a substantial difference between the theoretical number of outcomes and the number observed experimentally. We use a Markov process to simulate the temporal evolution of the outcomes' distribution. Results are illustrated figure V.7. Rather than specifying an arbitrary numeric value for λ, we calculate the distribution as a function of λt. This substitution removes the distribution's dependency to λ. Even though the Polylox cassette has a large number of derived strands, their distribution is far from uniform. In order to compare distributions with different number of outcomes and different probabilities, we use the *true diversity* with exponent 1, which can be calculated as the exponential of the Shannon entropy [205]. The true diversity can be interpreted as the number of equiprobable outcomes that would yield an identical entropy. The Polylox true diversity peaks shortly after the beginning of the reaction, as the distribution fills the many derived transgenes, but quickly converges with time to about 9 equivalent equiprobable outcomes. Even at its peak, the true diversity is far from the total 1866890 derived strands.



lox sites	inequalities	systems	solvable systems	*lox* types	transgenes	transgenes after normalization
2	$\binom{1}{1}$	$2^1 = 2$	2	1	8	2
3	$\binom{2}{1} + \binom{2}{2}$	$2^3 = 8$	5	1	40	5
4	$\binom{3}{1} + \binom{3}{2} + \binom{3}{3}$	$2^7 = 128$	19	1	304	57
				2	1824	152
5	$\binom{4}{1} + \binom{4}{2} + \binom{4}{3}$	$2^{14} = 16384$	142	1	4544	568
				2	45440	4686
6	$\binom{5}{1} + \binom{5}{2} + \binom{5}{3}$	$2^{25} \approx 3 \times 10^7$	2099	1	134336	20990
				2	4701760	356830
				3	12090240	138534
7	$\binom{6}{1} + \binom{6}{2} + \binom{6}{3}$	$2^{41} \approx 2 \times 10^{12}$	39949	1	5113472	818448
				2	286354432	35500182
				3	1073829120	38467056
total					**1382275520**	**75307510**

Fig. V.6 *In order to account for the minimum recombination distance of lox pairs, we generate a second database. Unlike the first database, it considers transgenes with inter-lox distances small enough to prevent some of the recombinations. Therefore, the passive elements' length impacts the nature of the outcomes and their distribution. To find optimal transgenes, we generate every length assignment yielding a unique set of enabled and disabled lox recombinations, expressed as a system of inequalities (top left). Only a small portion of the systems can be solved (bottom). Nevertheless, since the number of solvable systems multiplies the number of transgenes, the second database is much larger than the first (top right).*

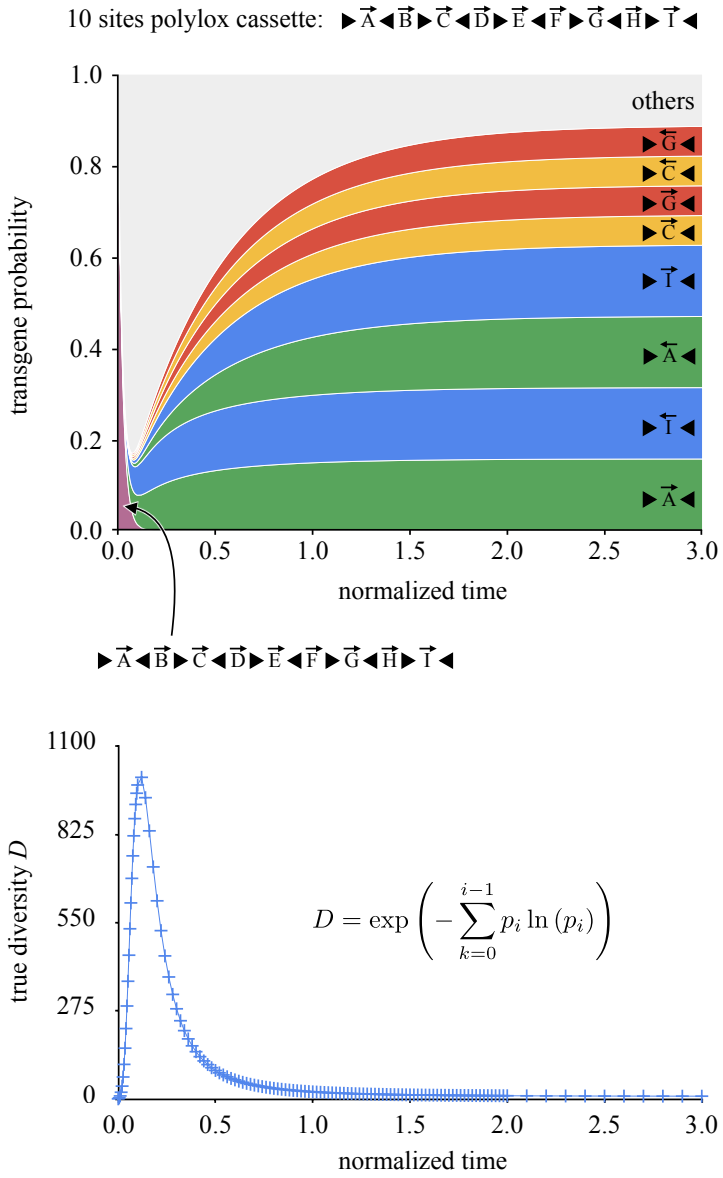


Fig. V.7 *The Polylox cassette introduced by Pei et al. [191] is designed for genetic barcoding: it aims to minimize collisions, hence its many outcomes (1866890). However, the outcomes are not equiprobably distributed. This figure shows the evolution of the outcomes' distribution over time (top), and the evolution of the true diversity over time (bottom). The latter can be interpreted as an equivalent number of equiprobable outcomes. After a relatively short time ($\frac{2.0}{\lambda}$), the true diversity is close to 9: despite a large number of outcomes, there is a substantial collision risk for the Polylox cassette.*

V.4.2 Database generation and analysis

The number of entries in the database grows exponentially with the number of *lox* sites. More *lox* sites correspond to larger Markov processes on average, resulting in longer and more memory-consuming weight calculations. Using a computer with a 500 GB RAM, we were able to calculate the outcomes' distribution of every transgene with up to nine *lox* sites and up to four *lox* sites types, without taking the minimum recombination distance into consideration. The generated data is held in the first database. In the second database, since each transgene is replicated numerous times to account for lengths assignments, we were able to generate every distribution for transgenes with up to seven *lox* sites and up to three *lox* sites types.

The database is designed to be searched using score functions. A score function assigns a numerical value to each transgene, with the highest value corresponding to the transgene best suited to a specific biological application. In order to provide insight on the contents of the database, we consider four relatively generic score functions: *number of outcomes*, *true diversity*, *number of outcomes (all irreversible)*, *true diversity (all outcomes irreversible)*. The first function associates a transgene with its number of outcome, whereas the second associates it with the true diversity of its outcomes' distribution. Their *irreversible* variants return the same values, except for transgenes with at least one reversible outcome, for which the score is zero.

Figure V.8 shows the maximum value of the four score functions for each number of *lox* sites and each number of *lox* sites types in the database. Both the number of combinations in the original transgene and the number of excisions decrease with the number of types. A compromise has to be reach to maximize outcomes, as the number of outcomes increases with the number of combinations but decreases with the number of excisions. The compromise shifts towards more types as the number of sites grows, or if only transgenes with irreversible outcomes are considered. The optimal number of types for a given number of sites is identical with respect to the *number of outcomes* and *true diversity* score functions.

Figure V.9 shows the same score functions as figure V.8 applied to the second database, which takes minimum recombination distances into account. The number of excisions does not increase when fewer types are used, since excisions can be prevented with well-chosen inter-site distances: single-type transgenes outperform transgenes with multiple types in the second database. For the same reason, using distances makes generating many irreversible outcomes easier.

V.5 Conclusion and discussion

This work introduced a novel model for Cre-*lox* recombinations. Unlike existing approaches, it does not rely on brute-force simulations, and does not assume that the dynamics of excisions are slow compared to the dynamics of inversions. This model is validated against existing biological experiments. We use its predictive capabilities to fill exhaustive databases of transgenes with up to four *lox* sites types and nine *lox* sites. The overall analysis of the databases' content gives insight on the relationship between a transgene's composition and the properties of its outcomes.

Our model is based on two hypothesis: Cre-*lox* recombinations are first order chemical

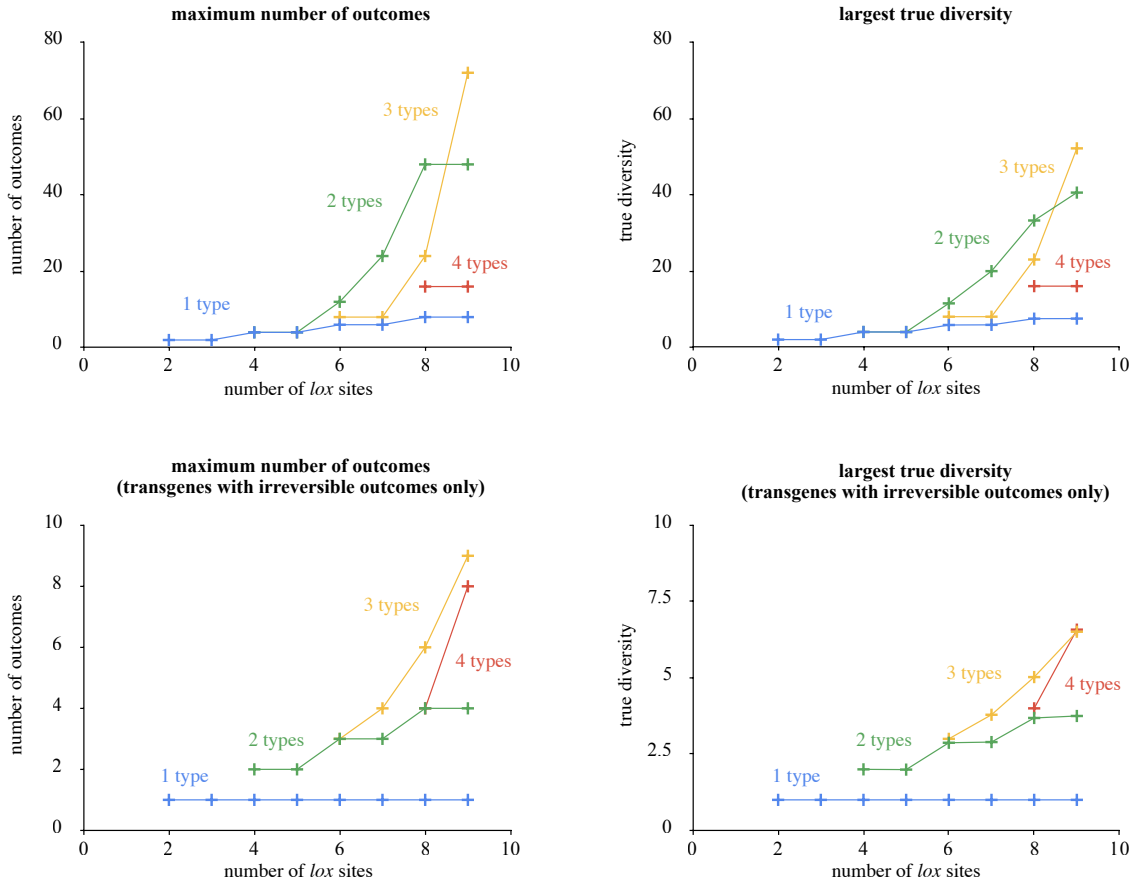


Fig. V.8 *We search the first database (which does not assume a minimum recombination distance) to find the transgenes with the largest number of outcomes. We retain one transgene for each number of lox sites (two to nine) and number of lox types (one to four). The four graphs correspond to four sets of constraints on the outcomes and their distribution, which are related to the constraints of real world problems.*

reactions, and a transgene does not bear any mark of its past mutations. These hypothesis are required to model the derived transgenes as the states of a Markov process. The first hypothesis is consistent with experimental observations [206]. The second hypothesis, on the other hand, is not easy to validate directly: memory effects could arise from phenomenons that cannot be easily measured, such as DNA folding or non-uniform Cre distribution in space. Nevertheless, the experimental validation of our model's results can be seen as an indirect validation of this hypothesis. A third hypothesis is added to generate the second database, which accounts for a minimum recombination distance. We assume that the recombination rate is constant if the inter-*lox* distance is larger than 82 base pairs, and zero otherwise. Further experiments have to be performed to quantify errors caused by this simplifying assumption.

The existing experiments used to validate the model rely on relatively simple Cre-*lox* transgenes. Future work will be conducted to assess the predictive capabilities of the

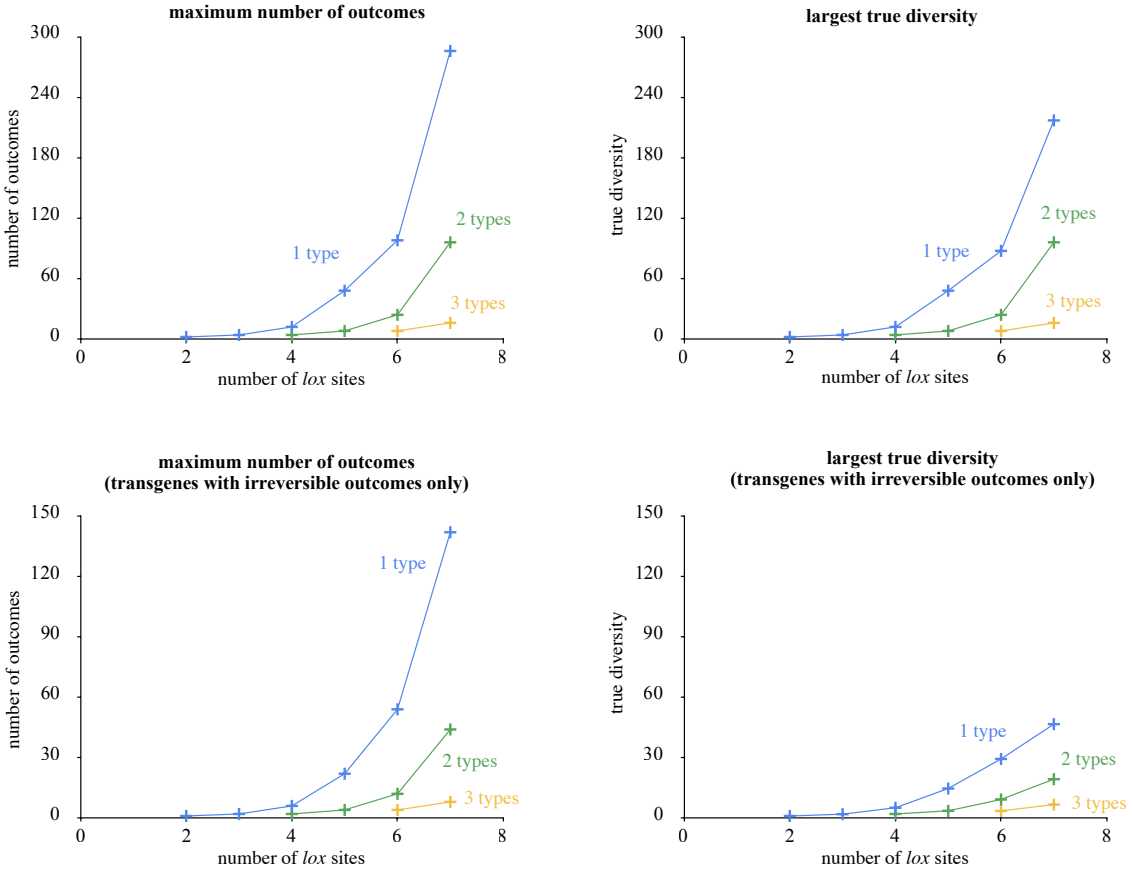


Fig. V.9 *We search the second database (which accounts for a minimum recombination distance, and contains every possible passive elements' length repartition) with the same method used to search the first one. However, the second database only contains transgenes with two to seven sites, and one to three types. The number of outcomes that can be obtained when accounting for recombination distances much larger regardless the constraints on the outcomes. Nevertheless, it assumes that the passive elements' number of base pairs can be chosen arbitrarily.*

model, by performing experiments on novel and more complex transgenes found using the database.

Markov processes have been extensively studied, and a wide variety of tools and methods can used to extract information from a model. For example, it is possible to compute the expected time before absorption, that is the average time before a transgene is mutated to an outcome. The variance on the expected time to absorption or the probability to visit a specific state can be calculated as well. For complex chains, the sparse matrix inversion used to calculate the outcomes' distribution requires very large amounts of memory and computational power. The *Markov Chain Monte-Carlo* method, another result of the theoretical work on Markov processes, provides an estimate of the results with smaller computational requirements.

This work approaches the problem of transgene design with a brute-force strategy, as usual string metrics, such as the Levenshtein distance, yield little regularity for the considered score functions. Though this approach produces results, it scales poorly to longer transgenes. Additional research is needed to better understand the transgenes space' topology, which is required to identify optimal transgenes with a large number of *lox* sites.

Chapter VI

Discussion

VI.1 Conclusion

This thesis set out to explore neuromorphic approaches to algorithms and vision sensors design, and their potential applications to biology. It lead to the development of methods, tools and devices contributing to neuromorphic engineering, with applications in psychophysics and biology. We introduced a software framework to implement event-based computer vision algorithms, since existing solutions could not be easily extended to handle events with arbitrary types. Two experimental devices leveraging the framework were developed: a three-chip event-based color camera and a psychophysics setup. Both devices feature a high temporal resolution inspired by precise-timing biological models. The results obtained with the color sensor show promise to automate the segmentation of biological samples. Within this context, we formulated a mathematical model for the biological engineering method *Brainbow*.

The presented framework leverages event-driven programming constructs, resulting in better algorithms semantics. Performance is measured with comparative benchmarks involving every other C/C++ framework for event-based computer vision. These benchmarks, to our knowledge the first of their kind, make it possible to quantify the role of software design choices in implementation performance. Moreover, they show that the presented framework outperforms other solutions in both speed and latency. The efficiency and speed of CPU implementations is important to neuromorphic engineering for two reasons. On the one hand, they provide a fair comparison to neuromorphic hardware, essential to estimate possible gains in power consumption and speed. On the other hand, they enable applications of neuromorphic sensors using conventional, widespread hardware.

Unlike other event-based color sensors, the camera presented in this work can perform event-driven absolute color measurements. We demonstrate its operation with an application to object tracking using only color cues and a simple event-driven algorithm. If performing absolute measurements yields good results, it also increases latency. § VI.4 questions the necessity of such measurements and proposes directions for future event-based color sensors.

Visual psychophysics help to pinpoint important features of the human visual system. This information plays a major role in the design of neuromorphic systems, notably

cameras. In turn, neuromorphic engineering methods can improve psychophysics setups, enabling novel experiments. The high-speed setup developed in this thesis is structured like an event-based vision system, with two machines: a real-time computer responsible for assigning precise timestamps to events and a non-real-time computer running algorithms. Recent works have shown that the human eye is sensitive to frequencies much larger than 60 Hz [174], the usual estimate for the *flicker fusion rate*. Our setup will be used to study the relationship between framerate and perception in primates. Conventional cameras cannot capture information faster than a few hundred hertz in real-time (very high-speed cameras are expensive and cannot record for an extended period of time). Showing that higher frequencies are used by biological systems, and improve their reliability, is a strong argument in favor of event-driven cameras.

Since neuromorphic engineering draws its inspiration from neuroscience and biology, its contributions to these research fields leads to synergies. Among biological methods that rely on engineering, *Brainbow* faces specific challenges: the images it generates have a high dynamic range, and are difficult to segment automatically. Event-driven cameras and algorithms are good candidates to overcome these issues, in view of the cameras' high dynamic range and the algorithms' unique approach to computer vision. The limited sensitivity of the ATIS in low light conditions prevented us from using our three-chip, event-based camera on *Brainbow* samples. Nevertheless, we formulated a mathematical model that will help identify novel transgenes to improve the method's performance, thus preparing the ground for novel event-based sensors with higher sensitivity. Future experiments will use Brainbow to study the retina, extending our understanding of its development and underlying mechanisms. The new knowledge may, in turn, result in advances in silicon retina design.

VI.2 Neuromorphic engineering: build brains to understand them

Neuromorphic engineering aims to build computers and sensory systems mimicking their biological counterparts, down to the transistor level. This approach has two goals: to understand brains by trying to build them, and to copy what nature came up with to improve technology.

Neuroscience aims to describe and understand the brain, however several shortcomings of its observational approach have been recently stressed. Notably, it has been shown that current neuroscience methods fail to provide a complete explanation of a CPU [207], even though they are able to extract valuable information. Elowitz et al. [208] argue that biologists and engineers have complementary approaches, as the former try to reverse engineer biological systems, whereas the latter aim to build them. The comment regards biological engineering, however it can be extended to neuromorphic engineering. The idea that, through building, neuromorphic engineering acts as explanatory neuroscience can be traced backed to the early days of the field [209]. The psychophysics setup presented in this thesis falls within this context. Algorithms for event-based cameras are an operational model for the processing of data with a high temporal resolution, such as the signals generated by the human eye, and motivated novel research to understand our visual system.

Neuromorphic engineering also aims to improve technology, noting the high perfor-

mance of biological systems compared to their artificial counterparts in terms of robustness and power consumption. The camera and framework introduced in this thesis are examples of this approach. Besides continuing advances in neuroscience and VLSI design, neuromorphic engineering benefits from a favourable context: the slowdown of Moore's law is an opportunity for new approaches to computation [210]. Neuromorphic engineering's potential to surpass the conventional approaches to computer design and artificial intelligence attracted the interest of large industrial companies: IBM and Intel both developed neuromorphic chips. Moreover, numerous startups arose from neuromorphic engineering, perpetuating Carver Mead's approach to research [211].

VI.3 Neuromorphic or event-based?

Neuromorphic engineering is a form of bioinspiration: nature is used as a source of inspiration to invent and improve technology. Bioinspired methods and techniques do not aim to perfectly replicate biology, but to understand and harness the underlying principles brought into play by nature. Thus, research fields with an interest aligned with that of biology can drift away and follow their own path. For example, perceptrons or convolutional neural networks, originally based on simple, rate-based neuron models, have grown to a research field of their own. Neuron layers in these models are nowadays abstracted as tensor operations and non-linear transforms, while novel biological experiments question rate models.

The neuromorphic silicon retinas were primarily designed as electronic models of the eye. However, the addition of arbiters and USB connectors, in the late 2000s, made them compatible with CPUs and lead to event-based cameras. Ongoing investigations aim to identify event-based algorithms to process the data generated by these cameras. One can wonder if event-based computer vision should try to closely mimic biological systems, or evolve on its own as an engineering method. The framework presented in this work achieves a compromise: biological systems are used as an overall guideline rather than a strong constraint. Notably, event-based calculations are encouraged, for they should translate to future neuromorphic hardware while yielding good performance on conventional architectures. The resulting methods, besides a model for the human visual system, represent a whole new approach to computer vision, with two notable features.

The first feature - and most surprising with regard to conventional computer vision - is the algorithms' increasing robustness with speed. Conventional computer vision applied to videos generally requires a trade-off between speed and robustness. The latter can be increased at the expense of simplicity, thus algorithms have to be executed at a reduced framerate if data is to be processed in real-time. On the other hand, the high temporal resolution of event-based data enables powerful simplifying assumptions. For example, a tracking algorithm may assume that objects move at most by one pixel between samples. The resulting algorithm is not only robust; it also features a very low latency, and requires little computational power. In this regard, events can be seen as a means to an end - data compression in order to go fast - rather than a fundamental calculation needed to extract information.

The second distinguishing feature of event-driven algorithms is the fundamentally spatio-temporal signal they leverage. Conventional video computer vision algorithms use

spatial algorithms designed for static images on every frame: time is treated as an extension to space. Event-driven algorithms can naturally express spatio-temporal features using a three-dimensional point cloud to represent data, with two space dimensions and one time dimension. The algorithms introduced in this thesis to manipulate color events extend this representation to six dimensions: the three aforementioned, and three color dimensions. Even though only two color dimensions are used in our work, in order to demonstrate the possibility to track an object with only color cues, it paves the way for color-spatio-temporal features. It may be possible to link such extractors to the recent observations of the human visual cortex that show an inextricable link between color and form processing [36].

VI.4 Absolute color measurements

The relationship between human color sensing - based on three cone photoreceptors - and human color perception is not simple. The transformation from red, green and blue components to perceptually uniform color spaces, such as CIEL*a*b*, relies on absolute measurements. DVS-like event-based cameras do not provide absolute measurements, thus it is not possible to capture colors with a three-chip DVS or a Bayer DVS in a way that is consistent with human vision. The ATIS used in this work integrates luminance upon detecting a change, providing event-based absolute measurements at the expanse of latency.

Humans' ability to differentiate colors decreases with spatial distance [212]. This property of our visual system is compatible with local color processing circuits identified in the retina. It is possible that most of the color information sent to the brain by the retina corresponds to detections of pre-processed spatial color contrast. The first silicon retina, as well as most silicon retinas invented in the 1990s and 2000s, detect spatial contrast using resistive layers. This is true of silicon retinas designed for robust color detection as well [82]. If research on spatial silicon retinas continues today, little focus is given to algorithms dedicated to process their output. Neuromorphic algorithms running on CPUs generally target temporal-contrast sensors [213] - such as the DVS or ATIS - for they are more widespread, less noisy, and easier to interfaces with. The wide variety of spatial-contrast sensors, often built in very limited series, complicates the estimation of their contribution to computer vision compared to temporal-contrast detectors. Even though many event-based pixels have been designed - including pixels performing color transformations, such as the *cDVS*'s pixels - only a handful have been upgraded to full-fledged cameras, for this process is both expensive and time consuming.

VI.5 Simulation

Neuromorphic engineering aims to develop both novel hardware and novel algorithms. Since both tasks are complex, hybrid approaches have been encouraged to separate the problems: analog neuromorphic sensors are associated with conventional digital computers. The development of such systems has been mostly driven by sensor availability: event-based algorithms are based on the premise that visual information is acquired by existing sensors. Yet, the analog-to-digital conversions performed by vision sensors, frame-

based and event-based alike, are neither equivalent nor reversible. Different silicon retinas generate fundamentally different digital information, with a major impact on the associated algorithms' performance and complexity. Novel, original silicon retinas may see the light of day by driving their development from a computer vision perspective, to meet the needs of algorithms. However, this strategy has undergone little exploration, in part because analog hardware development is complex and expensive.

We argue that the simulation of silicon retinas can provide new and original answers to the problem. Simulations are used extensively in biology and neuroscience to model the retina [214], with applications to silicon retinas design [215]. They also play a key role in neuromorphic hardware development to predict an electronic circuit's behaviour, though caution is advised in the analysis of their results [216]. Higher level simulations help study existing neuromorphic cameras, using computer-generated imagery [217] or high-speed frame-based cameras [218]. Even though they require large amounts of computational power, they provide great flexibility: sensors with arbitrary spatial and temporal resolution can be emulated. Different sensing strategies (such as frame-based and event-based) can be emulated with the same visual scene, enabling rigorous comparisons. Moreover, artificial frames generated from a 3D model come with a perfect ground truth, extremely valuable to algorithms benchmarks. Little research has been carried out on the use of high-level simulation to explore new camera designs. In a future work, we plan to explore computer-vision-driven sensors simulations, in order to identify analog operations that serve digital algorithms. The results may be used as a guideline to develop new event-based silicon retinas.

Appendix A
Event Stream file format

A.1 Containers

Multiple file formats co-exist to represent events generated by a camera. This appendix presents considerations to improve over these formats, and makes the first step in the suggested direction with a novel specification. This specification is used by the *sepia* library.

The C++ frameworks mentioned in this paper use the *dat* file format (*kAER*) or the *aerdat* format (*jAER*, *cAER* and *Dynamic Vision Systems*). The *dat* format has no official specification, and thus is hard to use without Prophesee's proprietary libraries. The *aerdat* fornat uses packets mixing frames and events, reflecting the output format of the DAVIS. Packets make the generation of artificial data more complex, since encapsulation in packets of arbitrary length is mandatory. Moreover, the format does not benefit from state-of-the-art frame-based encoding schemes, such as HEVC or VP9 [219].

Conventional multimedia files use containers, defined as a format wrapper holding common meta-data (title, author, creation date...) and encapsulating data streams. Each stream is encoded in a data format describing a specific type of information. This organization is modular and simplifies the design of data formats, as they do not need to account for meta-data.

A container for event-based data would wrap - among others - streams of visual events, IMU events, cochlea events and frames. The latter would take advantage of existing formats for frame-based data. Potential compression algorithms for event streams would progressively replace current events encoding without changing the container format.

We introduce a simple, well-defined data format called *Event Stream*, with extension **.es**. The format encodes only visual events, without packets. It cannot encode meta-data, apart from information required for proper decoding and processing (namely, the format version and the sensor width and height). Even though this data format can be used alone, it is meant to be included in a container format (which does not exist yet).

A.2 Event Stream

The data associated with a visual event can be represented as a tuple (x, y, t, p), where x and y are the pixel coordinates, t is the timestamp (generally expressed in microseconds)

and p is the polarity. The latter has an arbitrary size in the general case, and is a boolean for *DVS*-like polarity events.

Since the events are ordered, t increases from one event to the next, often by a small value. This property is leveraged by the *aerdat* format: the timestamp is encoded relatively to the packet beginning, reducing the number of bits required to encode it. Using a similar approach, a packet-free scheme can be devised:

- An event is encoded by k timestamp bits and 33 payload bits (16 bits for x, 16 bits for y and 1 bit for the polarity). Only timestamps in the range $[\![0, 2^k - 2]\!]$ are used.
- The special value $2^k - 1$ encodes a timestamp overflow: the decoding program must increment by $2^k - 1$ an offset variable initialized to zero and added to every timestamp. The 33 payload bits are not included in this case.

We refer to this scheme as absolute encoding. A variant, called relative encoding, consists in encoding each timestamps relatively to the previous one. The offset variable must be incremented with every event. Relative timestamp overflows are handled similarly to absolute timestamp overflows.

The throughput of the absolute and relative encoding schemes depends on k and the event stream content. Figure A.1 (top) shows the throughput of various streams as a function of k. The previously described *car* and *street* streams (table II.1), as well as the initial second of the *street* stream, are used. The beginning of the *street* stream contains very little activity (the average even rate is $22.2 \times 10^3\,\mathrm{s}^{-1}$), illustrating the schemes behaviour in this situation. The minimum throughput is obtained for surprisingly small values of k. Notably, the optimal value for k is one for the *car* stream. With this k, both schemes are identical and are equivalent to writing a binary one every time the clock advances by one microsecond, and a zero followed by the payload for each event. The performance variation between schemes is small compared to the throughput. Nevertheless, the relative scheme outperforms the absolute one. The difference increases with the activity, yielding a better compression when it is most needed.

The absolute encoding is sensitive to bit errors: they can create non-monotonic timestamps, thus negative time deltas. The latter are used in many algorithms, and negative values generally result in unexpected behaviors. Bit errors are less serious for the relative encoding: they result in small time distortions. Consequently, the *Event Stream* format specification is based on the relative scheme.

The schemes considered so far use a non-round number of bytes to encode each event. This approach complicates the implementation, since bytes are the fundamental type of most operating systems. Therefore, the *Event Stream* specification uses $k = 7$ (even though it is not the optimal value for high-activity streams) so that each event is encoded on 5 bytes. Offsets are encoded on one byte. The last bit is used to differentiate overflow bytes and reset bytes. The latter must be sent periodically if the encoding is used in a noisy environment. Upon reception, they reset the state machine illustrated figure A.2.

The *Event Stream* specification also supports ATIS events, color events and generic events (with an arbitrary payload associated with each timestamp). It is designed to be extended.

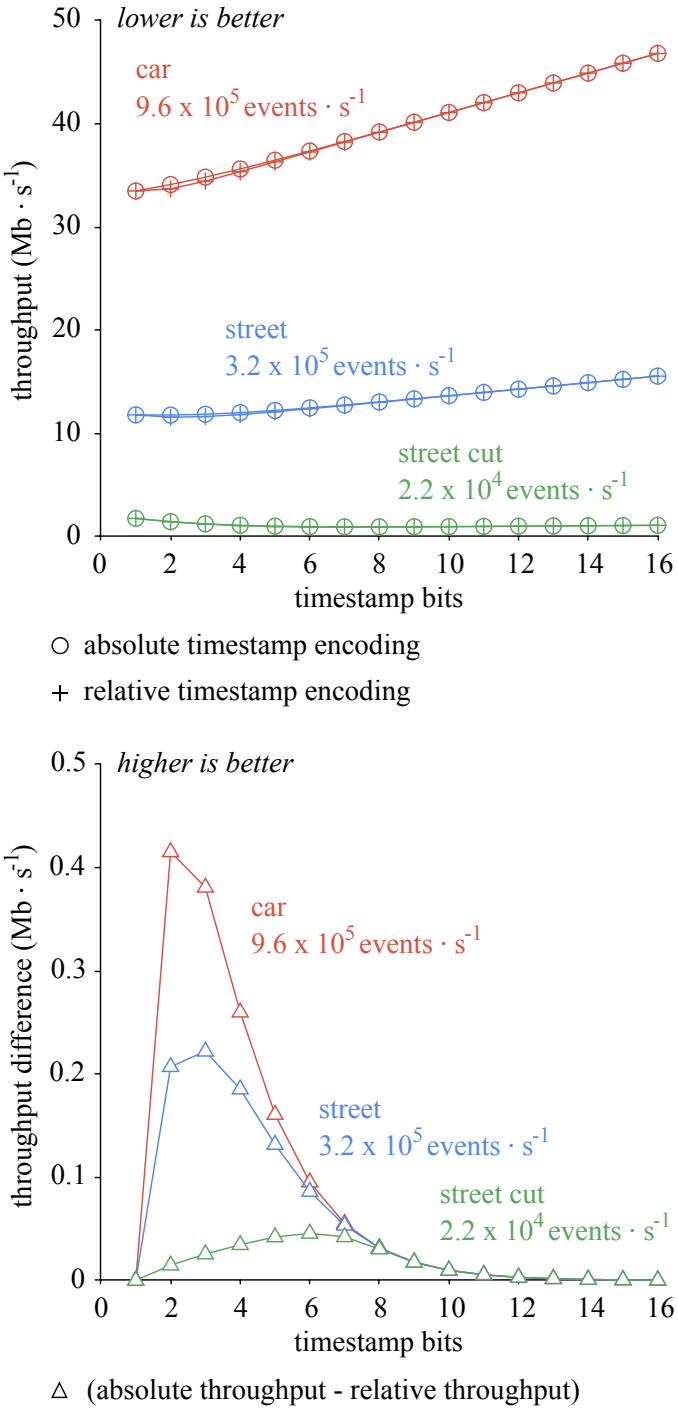


Fig. A.1 *The top graph plots the throughput (the number of bits required to encode the stream) as a function of the number of bits used to encode the timestamp. The absolute scheme is represented with circles, and the relative scheme with crosses. The bottom graph shows the throughput difference between schemes.*

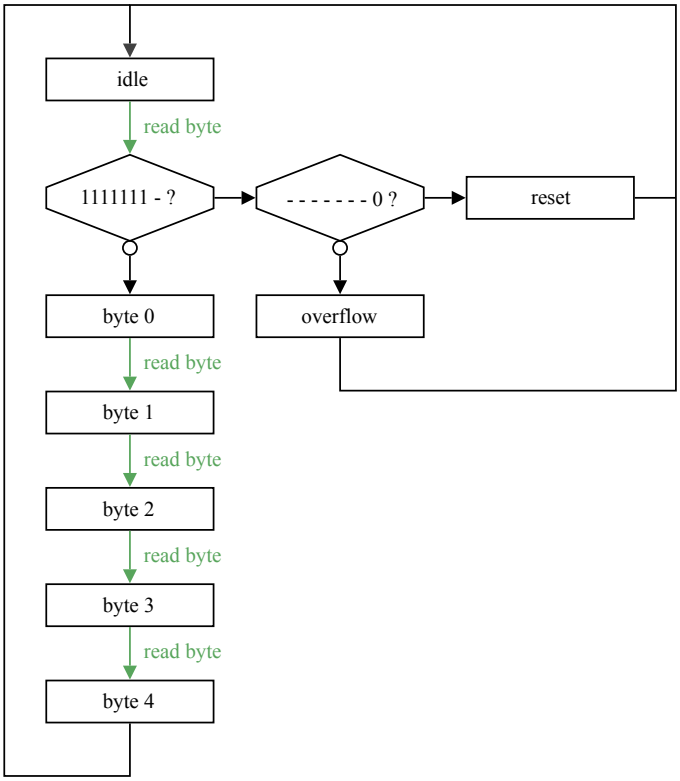


Fig. A.2 *This state machine describes the Event Stream specification for DVS-like events. The format is designed for file IO, but can be used over any serial communication channel, such as USB.*

Appendix B

3D printed parts for psychophysics

The code presented chapter IV and hosted in the Hummingbird repository is only concerned with the *DLP Lightcrafter 3000*. Experiments for which the other aspects of Hibiscus are not relevant need either a simple way to tilt the projector, or a basis to build an alternative box. The mechanical parts presented figure B.1, meant to be 3D printed, constitute a simplified version of the box presented in this work. The *stand* part is compatible with metric optical tables. The elbow part is a structural reinforcement to prevent heat-related distortions of PLA parts.

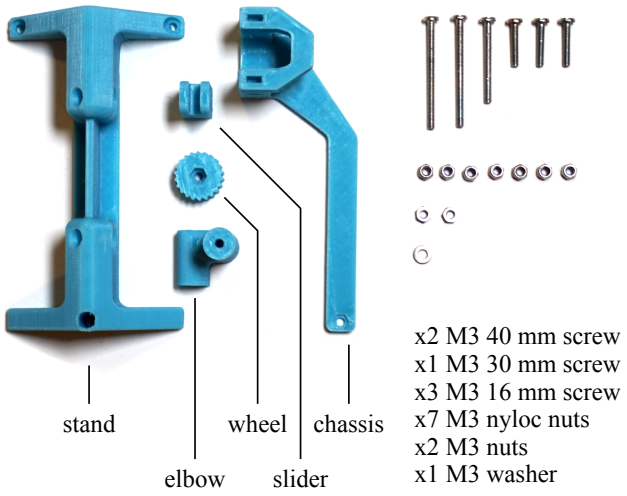


Fig. B.1 *The kit illustrated in this figure is a simplified version of the box presented in this box. It is intended for projects where a precisely timed response is not needed, or as a basis for an alternative box assembly. The parts designs are freely available online, and can be manufactured with most consumer 3D printers.*

Appendix C
Cre-*lox* algorithms

C.1 Building a transgene's Markov process

In order to build a transgene's Markov process, we first introduce the *build-inversion-set* function. Two transgenes are in the same inversion set if there is at least one sequence of inversions turning the first one into the second. The edges used by the algorithm are tuples of three elements: a source node, a target node and a weight.

The function *inversions* (respectively *excisions*) generates the set of transgenes obtained from every possible inversion (respectively excision) of the transgene passed as argument. The function *insert-or-sum* adds the edge given as second argument to the set given as first argument. If an edge with the same source and target is already in the set, the weights are summed. The function *pop* removes one of the given set's elements and returns it.

The *build-process* function generating a transgene's Markov process can be expressed using the function *build-inversion-set*.

C.2 Reducing a process

The Markov process associated with a transgene can be reduced by merging the absorbing sets into absorbing states. The *reduce-process* function takes as input the output of the *build-process* function described appendix C.1.

The function *insert-or-sum* is defined as in appendix C.1. The function *absorbing-set-to-node* associates each absorbing set from the original Markov process to a node of the reduced Markov process. The function *node-to-absorbing-set* associates each absorbing node of the original process with its absorbing set.

C.3 Generating repartitions

The *repartition* function generates possible types repartitions from a number of *lox* sites and a number of *lox* sites types. A repartition assigns a number of elements to each type. This function is used by by the database generation process, and takes part in determining every possible transgene with a given number of sites. The function *concatenate* creates a new ordered list by appending the elements of its second argument to the first.

```
Function build_inversion_set(transgene)
    nodes ← {transgene}
    edges ← {}
    external_nodes ← {}
    external_edges ← {}
    transgenes_to_handle ← {transgene}
    while transgenes_to_handle ≠ ∅ do
        transgene ← pop(transgenes_to_handle)
        number_of_recombinations ← 0
        foreach inverted_transgene of inversions(transgene) do
            number_of_recombinations ← number_of_recombinations + 1
            if inverted_transgene ∉ nodes then
                transgenes_to_handle
                  ← transgenes_to_handle ∪ {inverted_transgene}
            end
            nodes ← nodes ∪ {inverted_transgene}
            insert_or_sum(edges, (transgene, inverted_transgene, 1))
        end
        foreach excised_transgene of excisions(transgene) do
            number_of_recombinations ← number_of_recombinations + 1
            external_nodes ← external_nodes ∪ {excised_transgene}
            insert_or_sum(edges, (transgene, excised_transgene, 1))
        end
        divide the edges' weights by number_of_recombinations
    end
    return (nodes, edges, external_nodes, external_edges)
```

C.4 Normalizing a transgene

The database generation process yields transgenes with the same Markov processes, though their sites are different. The following normalization process merges such transgenes to reduce the number of required outcomes' distributions calculations to generate an exhaustive database.

1. We generate a copy of the transgene by reading it backwards (the sites positions and directions are inverted).

2. For both the original transgene and the backwards copy, we rename and re-orient the elements so that the types appear in order and so that the first site of a type is right-oriented.

3. Among the two renamed transgenes, the one that comes first with respect to the lexicographic order is the normalized version of the initial transgene.

As an example, the transgene ◀ ▷ ◁ ◁ ▶ becomes ◀ ▷ ▷ ◁ ▶ when read backwards (DNA strands bewteen *lox* sites are omitted, as they do not take part in the normalization

```
Function build_process(transgene)
    process_nodes ← {transgene}
    process_edges ← ∅
    absorbing_sets ← ∅
    transgenes_to_handle ← {transgene}
    handled_Transgenes ← {transgene}
    while transgenes_to_handle ≠ ∅ do
        transgene ← pop(transgenes_to_handle)
        if transgene ∈ handled_transgenes then
            continue
        end
        (nodes, edges, external_nodes, external_edges) ←
         build_inversion_set(transgene)
        process_nodes ← process_nodes ∪ nodes
        process_edges ← process_edges ∪ edges
        handled_Transgenes ← handled_Transgenes ∪ nodes
        if external_nodes = ∅ then
            absorbing_sets ← absorbing_sets ∪ {nodes}
        else
            transgenes_to_handle
              ← transgenes_to_handle ∪ (external_nodes \ process_nodes)
            process_nodes ← process_nodes ∪ external_nodes
            process_edges ← process_edges ∪ external_edges
        end
    end
    return (process_nodes, process_edges, absorbing_sets)
```

process). Assuming that ▷ has a type index smaller than ▶, the renamed and reoriented versions of these transgenes are ▷ ▶◀◀ ◁ and ▷ ▶▶◀ ◁. Since ▷ ▶▶◀ ◁ comes before ▷ ▶◀◀ ◁ in lexicographic order, the normalized version of ◀ ▷ ◁ ◁ ▶ is ▷ ▶▶◀ ◁.

```
Function reduce_process(process_nodes, process_edges, absorbing_sets)
    absorbing_nodes ← ⋃ (set ∈ absorbing_sets) set
    nodes ← process_nodes \ absorbing_nodes
    foreach set of absorbing_sets do
        nodes ← nodes ∪ {absorbing_set_to_node(set)}
    end
    edges ← ∅
    foreach edge of process_edges do
        if edge.source ∉ absorbing_nodes then
            if edge.target ∈ absorbing_nodes then
                target_absorbing_set ← node_to_absorbing_set(edge.target)
                new_edge_target ←
                 absorbing_set_to_node(target_absorbing_set)
                insert_or_sum(edges,
                 (edge.source, new_edge_target, edge.weight))
            else
                edges ← edges ∪ {edge}
            end
        end
    end
    return (nodes, edges)
```

```
Function repartitions(number_of_lox_sites, number_of_lox_sites_types,
 minimum_sites_by_type = 2)
    if number_of_lox_sites_types × minimum_sites_by_type >
     number_of_lox_sites then
        return ∅
    end
    if number_of_lox_sites_types = 1 then
        return {[number_of_lox_sites]}
    end
    cumulated_repartitions ← ∅
    pivot ← minimum_sites_by_type
    loop
        number_of_lox_sites_left ← number_of_lox_sites − pivot
        number_of_types_left ← number_of_lox_sites_types − 1
        subrepartitions ← repartitions(number_of_lox_sites_left,
         number_of_types_left, pivot)
        if subrepartitions = ∅ then
            break
        end
        foreach subrepartition of subrepartitions do
            subrepartition ← concatenate(subrepartition, [pivot])
            cumulated_repartitions ← cumulated_repartitions ∪ {subrepartition}
        end
        pivot ← pivot + 1
    end
    return cumulated_repartitions
```

Bibliography

[1] D. Burton; *The History of Mathematics: An Introduction* (McGraw-Hill) (2007); ISBN 9780073051895; URL https://books.google.com/books?id=VDlGAQAAIAAJ. xi

[2] G. Galilei & S. Drake; *Discoveries and Opinions of Galileo: Including The Starry Messenger (1610), Letter to the Grand Duchess Christina (1615), and Excerpts from Letters on Sunspots (1613), The Assayer (1623)*; Anchor books (Doubleday) (1957); ISBN 9780385092395; URL https://archive.org/details/B-001-001-741. xi

[3] A. M. Turing; "On Computable Numbers, with an Application to the Entscheidungsproblem"; Proceedings of the London Mathematical Society **s2-42**, p. 230–265 (1936); ISSN 0024-6115; URL http://dx.doi.org/10.1112/plms/s2-42.1.230. xi

[4] A. M. Turing; "I.—COMPUTING MACHINERY AND INTELLIGENCE"; Mind **LIX**, p. 433–460 (1950); ISSN 1460-2113; URL http://dx.doi.org/10.1093/mind/lix.236.433. xi

[5] M. Campbell, A. Hoane & F.-h. Hsu; "Deep Blue"; Artificial Intelligence **134**, p. 57–83 (2002); ISSN 0004-3702; URL http://dx.doi.org/10.1016/s0004-3702(01)00129-1. xi

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot & et al.; "Mastering the game of Go with deep neural networks and tree search"; Nature **529**, p. 484–489 (2016); ISSN 1476-4687; URL http://dx.doi.org/10.1038/nature16961. xi

[7] S. Papert; "The Summer Vision Project"; http://hdl.handle.net/1721.1/6125 (1966). xi

[8] E. Kandel, T. Jessell, J. Schwartz, S. Siegelbaum & A. Hudspeth; *Principles of Neural Science, Fifth Edition*; Principles of Neural Science (McGraw-Hill Education) (2013); ISBN 9780071390118; URL https://books.google.fr/books?id=s64z-LdAIsEC. xi, 2

[9] G. Indiveri & T. K. Horiuchi; "Frontiers in Neuromorphic Engineering"; Frontiers in Neuroscience **5** (2011); ISSN 1662-4548; URL http://dx.doi.org/10.3389/fnins.2011.00118. xi

[10] F. C. Morabito, A. G. Andreou & E. Chicca; "Neuromorphic Engineering: From Neural Systems to Brain-Like Engineered Systems"; Neural Networks **45**, p. 1–3 (2013); ISSN 0893-6080; URL http://dx.doi.org/10.1016/j.neunet.2013.07.001. xi

[11] A. Vanarse, A. Osseiran & A. Rassau; "A Review of Current Neuromorphic Approaches for Vision, Auditory, and Olfactory Sensors"; Frontiers in Neuroscience **10** (2016); ISSN 1662-453X; URL http://dx.doi.org/10.3389/fnins.2016.00115. xi, 7

[12] R. A. Nawrocki, R. M. Voyles & S. E. Shaheen; "A Mini Review of Neuromorphic Architectures and Implementations"; IEEE Transactions on Electron Devices **63**, p. 3819–3829 (2016); ISSN 1557-9646; URL http://dx.doi.org/10.1109/ted.2016.2598413. xi

[13] P. Lichtsteiner, C. Posch & T. Delbruck; "A 128×128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor"; IEEE Journal of Solid-State Circuits **43**, p. 566–576 (2008); ISSN 0018-9200; URL http://dx.doi.org/10.1109/jssc.2007.914337. xi, 7

[14] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi & M. Srinivasan; "Asynchronous frameless event-based optical flow"; Neural Networks **27**, p. 32–37 (2012); ISSN 0893-6080; URL http://dx.doi.org/10.1016/j.neunet.2011.11.001. xii, 10

[15] F. Galluppi, K. Brohan, S. Davidson, T. Serrano-Gotarredona, J.-A. P. Carrasco, B. Linares-Barranco & S. Furber; "A Real-Time, Event-Driven Neuromorphic System for Goal-Directed Attentional Selection"; Lecture Notes in Computer Science p. 226–233 (2012); ISSN 1611-3349; URL http://dx.doi.org/10.1007/978-3-642-34481-7_28. xii, 10

[16] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat & R. Benosman; "Asynchronous Event-Based Multikernel Algorithm for High-Speed Visual Features Tracking"; IEEE Transactions on Neural Networks and Learning Systems **26**, p. 1710–1720 (2015); ISSN 2162-2388; URL http://dx.doi.org/10.1109/tnnls.2014.2352401. xii, 10, 42

[17] E. C. Carter (Editor); *Color Research and Application*; volume 44 (Wiley) (2019). xii

[18] S. R. Vantaram & E. Saber; "Survey of contemporary trends in color image segmentation"; Journal of Electronic Imaging **21**, pp. 040901–1 (2012); ISSN 1017-9909; URL http://dx.doi.org/10.1117/1.jei.21.4.040901. xii, 13, 41

[19] A. Voulodimos, N. Doulamis, A. Doulamis & E. Protopapadakis; "Deep Learning for Computer Vision: A Brief Review"; Computational Intelligence and Neuroscience **2018**, p. 1–13 (2018); ISSN 1687-5273; URL http://dx.doi.org/10.1155/2018/7068349. xii

[20] C. Posch, D. Matolin & R. Wohlgenannt; "An asynchronous time-based image sensor"; 2008 IEEE International Symposium on Circuits and Systems (2008); URL http://dx.doi.org/10.1109/iscas.2008.4541871. xii

[21] J. Livet, T. A. Weissman, H. Kang, R. W. Draft, J. Lu, R. A. Bennis, J. R. Sanes & J. W. Lichtman; "Transgenic strategies for combinatorial expression of fluorescent proteins in the nervous system"; Nature **450**, p. 56–62 (2007); ISSN 1476-4687; URL http://dx.doi.org/10.1038/nature06293. xii, 16, 80, 81

[22] N. J. Dominy & P. W. Lucas; "Ecological importance of trichromatic vision to primates"; Nature **410**, pp. 363–366 (2001); ISSN 0028-0836; URL http://dx.doi.org/10.1038/35066567. 1

[23] A. Trémeau, S. Tominaga & K. N. Plataniotis; "Color in Image and Video Processing: Most Recent Trends and Future Research Directions"; EURASIP Journal on Image and Video Processing **2008**, pp. 1–26 (2008); ISSN 1687-5281; URL http://dx.doi.org/10.1155/2008/581371. 1

[24] C. Bahlmann, Y. Zhu, V. Ramesh, M. Pellkofer & T. Koehler; "A system for traffic sign detection, tracking, and recognition using color, shape, and motion information"; in "IEEE Proceedings. Intelligent Vehicles Symposium, 2005. Las Vegas, NV, USA," pp. 255–260 (2005); ISSN 1931-0587; URL http://dx.doi.org/10.1109/IVS.2005.1505111. 1

[25] P. Kakumanu, S. Makrogiannis & N. Bourbakis; "A survey of skin-color modeling and detection methods"; Pattern Recognition **40**, pp. 1106–1122 (2007); ISSN 0031-3203; URL http://dx.doi.org/10.1016/j.patcog.2006.06.010. 1

[26] J. van de Weijer, T. Gevers & A. Bagdanov; "Boosting color saliency in image feature detection"; IEEE Transactions on Pattern Analysis and Machine Intelligence **28**, pp. 150–156 (2006); ISSN 0162-8828; URL http://dx.doi.org/10.1109/tpami.2006.3. 1

[27] J.-W. Hsieh, L.-C. Chen, S.-Y. Chen, D.-Y. Chen, S. Alghyaline & H.-F. Chiang; "Vehicle Color Classification Under Different Lighting Conditions Through Color Correction"; IEEE Sensors Journal **15**, pp. 971–983 (2015); ISSN 2379-9153; URL http://dx.doi.org/10.1109/jsen.2014.2358079. 1

[28] A. Valberg; *Light Vision Color* (Wiley) (2005); ISBN 9780470849026; URL https://books.google.fr/books?id=0oESifAi9ZsC. 1

[29] W. A. Thornton; "How strong metamerism disturbs color spaces"; Color Research & Application **23**, p. 402–407 (1998); ISSN 1520-6378; URL http://dx.doi.org/10.1002/(sici)1520-6378(199812)23:6<402::aid-col8>3.0.co;2-x. 1

[30] A. L. Stubbs & C. W. Stubbs; "Spectral discrimination in color blind animals via chromatic aberration and pupil shape"; Proceedings of the National Academy of Sciences **113**, p. 8206–8211 (2016); ISSN 1091-6490; URL http://dx.doi.org/10.1073/pnas.1524578113. 1

[31] F. H. Zaidi, J. T. Hull, S. N. Peirson, K. Wulff, D. Aeschbach, J. J. Gooley, G. C. Brainard, K. Gregory-Evans, J. F. Rizzo & C. A. Czeisler; "Short-Wavelength Light Sensitivity of Circadian, Pupillary, and Visual Awareness in Humans Lacking an Outer Retina"; Current Biology **17**, p. 2122–2128 (2007); ISSN 0960-9822; URL http://dx.doi.org/10.1016/j.cub.2007.11.034. 1

[32] J. L. Schnapf; "Photosensitivity of Primate Photoreceptors"; in "Advances in Photoreception: Proceedings of a Symposium on Frontiers of Visual Science.", (National Academies Press) (1990); ISBN 9780309042406; URL http://dx.doi.org/10.17226/1570. 1, 2

[33] K. Jameson & R. G. D'Andrade; "It's not really red, green, yellow, blue: an inquiry into perceptual color space"; Color Categories in Thought and Language p. 295–319 (1996); URL http://dx.doi.org/10.1017/cbo9780511519819.014. 2

[34] B. R. Conway, S. Chatterjee, G. D. Field, G. D. Horwitz, E. N. Johnson, K. Koida & K. Mancuso; "Advances in Color Science: From Retina to Behavior"; Journal of Neuroscience **30**, p. 14955–14963 (2010); ISSN 1529-2401; URL http://dx.doi.org/10.1523/jneurosci.4348-10.2010. 2

[35] B. R. Conway; "Color Vision, Cones, and Color-Coding in the Cortex"; The Neuroscientist **15**, p. 274–290 (2009); ISSN 1089-4098; URL http://dx.doi.org/10.1177/1073858408331369. 2

[36] R. Shapley & M. J. Hawken; "Color in the Cortex: single- and double-opponent cells"; Vision Research **51**, pp. 701–717 (2011); ISSN 0042-6989; URL http://dx.doi.org/10.1016/j.visres.2011.02.012. 3, 102

[37] I. Rentzeperis, A. R. Nikolaev, D. C. Kiper & C. van Leeuwen; "Distributed processing of color and form in the visual cortex"; Frontiers in Psychology **5** (2014); ISSN 1664-1078; URL http://dx.doi.org/10.3389/fpsyg.2014.00932. 3

[38] J. J. McCann; "Retinex at 50: color theory and spatial algorithms, a review"; Journal of Electronic Imaging **26**, p. 031204 (2017); ISSN 1017-9909; URL http://dx.doi.org/10.1117/1.jei.26.3.031204. 3

[39] S. K. Shevell & F. A. A. Kingdom; "Color in Complex Scenes"; Annual Review of Psychology **59**, p. 143–166 (2008); ISSN 1545-2085; URL http://dx.doi.org/10.1146/annurev.psych.59.103006.093619. 3

[40] C. Wootton; *A Practical Guide to Video and Audio Compression: From Sprockets and Rasters to Macro Blocks* (Focal Press) (2005); ISBN 0240806301; URL https://www.xarg.org/ref/a/0240806301/. 4

[41] G. A. Kindem; *The Demise of Kinemacolor: Technological, Legal, Economic, and Aesthetic Problems in Early Color Cinema History*; volume 20 (University of Texas Press, Society for Cinema & Media Studies) (1981); URL http://www.jstor.org/stable/1224830. 4

[42] G. A. Kindem; *Hollywood's Conversion to Color: The Technological, Economic and Aesthetic Factors*; volume 31 (University of Illinois Press) (1979); URL http://www.jstor.org/stable/20687473. 4

[43] L. M. Christine Fernandez-Maloigne, Frederique Robert-Inacio; *Digital Color: Acquisition, Perception, Coding and Rendering* (Wiley-ISTE) (2012); ISBN 1848213468; URL https://www.xarg.org/ref/a/1848213468/. 4

[44] R. A. Maschal Jr, S. S. Young, J. Reynolds, K. Krapels, J. Fanning & T. Corbin; "Review of Bayer pattern CFA demosaicing with new quality assessment algorithms"; Infrared Imaging Systems: Design, Analysis, Modeling, and Testing XXI (2010); URL http://dx.doi.org/10.1117/12.849314. 4

[45] R. Lenz; "Optoelectronic colored image converter"; https://patents.google.com/patent/US5877807 (1997). 4

[46] R. F. Lyon & P. M. Hubel; "Eyeing the camera: Into the next century"; in "in Proc. IS&T/SID 10th Color Imaging Conf., 2002," pp. 349–355 (2002). 4

[47] M. Vlachos, D. Skarlatos & P. Bodin; "Foveon vs Bayer: comparison of 3D reconstruction performances"; ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences **XLII-2/W9**, p. 755–761 (2019); ISSN 2194-9034; URL http://dx.doi.org/10.5194/isprs-archives-xlii-2-w9-755-2019. 4

[48] V. C. Coffey; "Multispectral Imaging Moves into the Mainstream"; Optics and Photonics News **23**, p. 18 (2012); ISSN 1541-3721; URL http://dx.doi.org/10.1364/opn.23.4.000018. 4

[49] N. Hagen & M. W. Kudenov; "Review of snapshot spectral imaging technologies"; Optical Engineering **52**, p. 090901 (2013); ISSN 0091-3286; URL http://dx.doi.org/10.1117/1.oe.52.9.090901. 4

[50] M. Dinguirard & P. N. Slater; "Calibration of Space-Multispectral Imaging Sensors"; Remote Sensing of Environment **68**, p. 194–205 (1999); ISSN 0034-4257; URL http://dx.doi.org/10.1016/s0034-4257(98)00111-4. 4

[51] R. M. Levenson & J. R. Mansfield; "Multispectral imaging in biology and medicine: Slices of life"; Cytometry Part A **69A**, p. 748–758 (2006); ISSN 1552-4930; URL http://dx.doi.org/10.1002/cyto.a.20319. 4

[52] J. Brauers, N. Schulte & T. Aach; "Multispectral Filter-Wheel Cameras: Geometric Distortion Model and Compensation Algorithms"; IEEE Transactions on Image Processing **17**, p. 2368–2380 (2008); ISSN 1057-7149; URL http://dx.doi.org/10.1109/tip.2008.2006605. 4

[53] L. Miao, H. Qi & W. Snyder; "A generic method for generating multispectral filter arrays"; 2004 International Conference on Image Processing, 2004. ICIP '04. (2004); URL http://dx.doi.org/10.1109/icip.2004.1421830. 4

[54] R. Gehrke & A. Greiwe; "Multispectral image capturing with Foveon sensors"; ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences **XL-1/W2**, p. 151–156 (2013); ISSN 1682-1777; URL http://dx.doi.org/10.5194/isprsarchives-xl-1-w2-151-2013. 4

[55] N. Ibraheem, M. Hasan, R. Z. Khan & P. K Mishra; "Understanding Color Models: A Review"; ARPN Journal of Science and Technology **2** (2012). 4

[56] T. Berk, A. Kaufman & L. Brownston; "A human factors study of color notation systems for computer graphics"; Communications of the ACM **25**, p. 547–550 (1982); ISSN 0001-0782; URL http://dx.doi.org/10.1145/358589.358606. 4

[57] A. Zeileis, K. Hornik & P. Murrell; "Escaping RGBland: Selecting colors for statistical graphics"; Computational Statistics & Data Analysis **53**, p. 3259–3270 (2009); ISSN 0167-9473; URL http://dx.doi.org/10.1016/j.csda.2008.11.033. 4

[58] A. Thwaites, C. Wingfield, E. Wieser, A. Soltan, W. D. Marslen-Wilson & I. Nimmo-Smith; "Entrainment to the CIECAM02 and CIELAB colour appearance models in the human cortex"; Vision Research **145**, p. 1–10 (2018); ISSN 0042-6989; URL http://dx.doi.org/10.1016/j.visres.2018.01.011. 4

[59] E. R. Fossum; "Active pixel sensors: are CCDs dinosaurs?" Charge-Coupled Devices and Solid State Optical Sensors III (1993); URL http://dx.doi.org/10.1117/12.148585. 6

[60] M. Meister & M. Tessier-Lavigne; "Low-Level Visual Processing: The Retina"; in "Principles of Neural Science, Fifth Edition," Principles of Neural Science (McGraw-Hill Education) (2013); ISBN 9780071390118; URL https://books.google.fr/books?id=s64z-LdAIsEC. 6

[61] M. A. Mahowald & C. Mead; "Silicon Retina"; in "Analog VLSI and Neural Systems," (Addison-Wesley) (1989); ISBN 0201059924. 6

[62] T. Delbruck; "Silicon retina with correlation-based, velocity-tuned pixels"; IEEE Transactions on Neural Networks **4**, pp. 529–541 (1993); ISSN 1045-9227; URL http://dx.doi.org/10.1109/72.217194. 6

[63] R. Etienne-Cummings & J. Van der Spiegel; "Neuromorphic vision sensors"; Sensors and Actuators A: Physical **56**, p. 19–29 (1996); ISSN 0924-4247; URL http://dx.doi.org/10.1016/0924-4247(96)01277-0. 6

[64] J. Krammer & C. Koch; "Pulse-based analog VLSI velocity sensors"; IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing **44**, p. 86–101 (1997); ISSN 1057-7130; URL http://dx.doi.org/10.1109/82.554431. 6

[65] T. Delbruck & C. Mead; "Adaptive photoreceptor with wide dynamic range"; Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94 (2002); URL http://dx.doi.org/10.1109/iscas.1994.409266. 6

[66] J. Kramer & G. Indiveri; "Neuromorphic vision sensors and preprocessors in system applications"; Advanced Focal Plane Arrays and Electronic Cameras II (1998); URL http://dx.doi.org/10.1117/12.324013. 7

[67] S.-C. Liu & T. Delbruck; "Neuromorphic sensory systems"; Current Opinion in Neurobiology **20**, p. 288–295 (2010); ISSN 0959-4388; URL http://dx.doi.org/10.1016/j.conb.2010.03.007. 7

[68] T. Delbruck, B. Linares-Barranco, E. Culurciello & C. Posch; "Activity-driven, event-based vision sensors"; Proceedings of 2010 IEEE International Symposium on Circuits and Systems (2010); URL http://dx.doi.org/10.1109/iscas.2010.5537149. 7, 10

[69] T. Delbruck & S.-C. Liu; "Event-based silicon retinas and cochleas"; Frontiers in Sensing p. 87–100 (2012); URL http://dx.doi.org/10.1007/978-3-211-99749-9_6. 7

[70] M. Mahowald; *An Analog VLSI System for Stereoscopic Vision* (Springer US, Boston, MA) (1994); ISBN 978-1-4615-2724-4. 7

[71] K. Boahen; "Point-to-point connectivity between neuromorphic chips using address events"; IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing **47**, p. 416–434 (2000); ISSN 1057-7130; URL http://dx.doi.org/10.1109/82.842110. 7, 9

[72] P.-F. Ruedi, P. Heim, F. Kaess, E. Grenet, F. Heitger, P.-Y. Burgi, S. Gyger & P. Nussbaum; "A 128 x 128 pixel 120-db dynamic-range vision-sensor chip for image contrast and orientation extraction"; IEEE Journal of Solid-State Circuits **38**, p. 2325–2333 (2003); ISSN 0018-9200; URL http://dx.doi.org/10.1109/jssc.2003.819169. 7

[73] R. Berner & T. Delbruck; "Event-Based Pixel Sensitive to Changes of Color and Brightness"; IEEE Transactions on Circuits and Systems I: Regular Papers **58**, p. 1581–1590 (2011); ISSN 1558-0806; URL http://dx.doi.org/10.1109/tcsi.2011.2157770. 7, 9, 51

[74] C. Posch, D. Matolin & R. Wohlgenannt; "A QVGA 143dB dynamic range asynchronous address-event PWM dynamic image sensor with lossless pixel-level video compression"; 2010 IEEE International Solid-State Circuits Conference - (ISSCC) (2010); URL http://dx.doi.org/10.1109/isscc.2010.5433973. 7, 9

[75] C. Brandli, R. Berner, M. Yang, S.-C. Liu & T. Delbruck; "A 240 × 180 130 dB 3 μs Latency Global Shutter Spatiotemporal Vision Sensor"; IEEE Journal of Solid-State Circuits **49**, p. 2333–2341 (2014); ISSN 1558-173X; URL http://dx.doi.org/10.1109/jssc.2014.2342715. 7

[76] J. A. Lenero-Bardallo, T. Serrano-Gotarredona & B. Linares-Barranco; "A 3.6 μs Latency Asynchronous Frame-Free Event-Driven Dynamic-Vision-Sensor"; IEEE Journal of Solid-State Circuits **46**, p. 1443–1455 (2011); ISSN 1558-173X; URL http://dx.doi.org/10.1109/jssc.2011.2118490. 7

[77] T. Serrano-Gotarredona & B. Linares-Barranco; "A 128 ×128 1.5% Contrast Sensitivity 0.9% FPN 3 μs Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers"; IEEE Journal of Solid-State Circuits **48**, p. 827–838 (2013); ISSN 1558-173X; URL http://dx.doi.org/10.1109/jssc.2012.2230553. 7

[78] M. Yang, S.-C. Liu & T. Delbruck; "A Dynamic Vision Sensor With 1% Temporal Contrast Sensitivity and In-Pixel Asynchronous Delta Modulator for Event Encoding"; IEEE Journal of Solid-State Circuits **50**, p. 2149–2160 (2015); ISSN 1558-173X; URL http://dx.doi.org/10.1109/jssc.2015.2425886. 7

[79] S. Mafrica, S. Godiot, M. Menouni, M. Boyron, F. Expert, R. Juston, N. Marchand, F. Ruffier & S. Viollet; "A bio-inspired analog silicon retina with Michaelis-Menten auto-adaptive pixels sensitive to small and large changes in light"; Optics Express **23**, p. 5614 (2015); ISSN 1094-4087; URL http://dx.doi.org/10.1364/oe.23.005614. 7

[80] B. Son, Y. Suh, S. Kim, H. Jung, J.-S. Kim, C. Shin, K. Park, K. Lee, J. Park, J. Woo & et al.; "4.1 A 640×480 dynamic vision sensor with a 9 μm pixel and 300 Meps address-event representation"; 2017 IEEE International Solid-State Circuits Conference (ISSCC) (2017); URL http://dx.doi.org/10.1109/isscc.2017.7870263. 7

[81] J. Huang, M. Guo & S. Chen; "A dynamic vision sensor with direct logarithmic output and full-frame picture-on-demand"; 2017 IEEE International Symposium on Circuits and Systems (ISCAS) (2017); URL http://dx.doi.org/10.1109/iscas.2017.8050546. 7

[82] K. Shimonomura & T. Yagi; "A silicon retina system for color constancy"; in "2010 World Automation Congress," pp. 1–5 (2010); ISSN 2154-4824. 9, 102

[83] Z. Fu, R. Mao, A. N. Cartwright & A. H. Titus; "Neuromorphic optical sensor chip with color change-intensity change disambiguation"; Nanoscale Imaging, Sensing, and Actuation for Biomedical Applications VII (2010); URL http://dx.doi.org/10.1117/12.851044. 9

[84] C. Li, C. Brandli, R. Berner, H. Liu, M. Yang, S.-C. Liu & T. Delbruck; "Design of an RGBW color VGA rolling and global shutter dynamic and active-pixel vision sensor"; 2015 IEEE International Symposium on Circuits and Systems (ISCAS) (2015); URL http://dx.doi.org/10.1109/iscas.2015.7168734. 9

[85] G. Taverni, D. Paul Moeys, C. Li, C. Cavaco, V. Motsnyi, D. San Segundo Bello & T. Delbruck; "Front and Back Illuminated Dynamic and Active Pixel Vision Sensors Comparison"; IEEE Transactions on Circuits and Systems II: Express Briefs **65**, p. 677–681 (2018); ISSN 1558-3791; URL http://dx.doi.org/10.1109/tcsii.2018.2824899. 9

[86] C. Scheerlinck, H. Rebecq, T. Stoffregen, N. Barnes, R. Mahony & D. Scaramuzza; "CED: Color Event Camera Dataset"; in "The IEEE Confer-

ence on Computer Vision and Pattern Recognition (CVPR) Workshops," (2019). 9, 15

[87] X. Guo, X. Qi & J. G. Harris; "A Time-to-First-Spike CMOS Image Sensor"; IEEE Sensors Journal **7**, p. 1165–1175 (2007); ISSN 1530-437X; URL http://dx.doi.org/10.1109/jsen.2007.900937. 9

[88] N. Wu; "Neuromorphic vision chips"; Science China Information Sciences **61** (2018); ISSN 1869-1919; URL http://dx.doi.org/10.1007/s11432-017-9303-0. 10

[89] D. G. Chen, D. Matolin, A. Bermak & C. Posch; "Pulse-Modulation Imaging—Review and Performance Analysis"; IEEE Transactions on Biomedical Circuits and Systems **5**, p. 64–82 (2011); ISSN 1940-9990; URL http://dx.doi.org/10.1109/tbcas.2010.2075929. 10

[90] R. Dominguez-Castro, S. Espejo, A. Rodriguez-Vazquez, R. Carmona, P. Foldesy, A. Zarandy, P. Szolgay, T. Sziranyi & T. Roska; "A 0.8 μm CMOS two-dimensional programmable mixed-signal focal-plane array processor with on-chip binary imaging and instructions storage"; IEEE Journal of Solid-State Circuits **32**, p. 1013–1026 (1997); ISSN 0018-9200; URL http://dx.doi.org/10.1109/4.597292. 10

[91] N. Massari, M. Gottardi, L. Gonzo, D. Stoppa & A. Simoni; "A CMOS Image Sensor With Programmable Pixel-Level Analog Processing"; IEEE Transactions on Neural Networks **16**, p. 1673–1684 (2005); ISSN 1045-9227; URL http://dx.doi.org/10.1109/tnn.2005.854369. 10

[92] P. Dudek & P. Hicks; "A general-purpose processor-per-pixel analog SIMD vision chip"; IEEE Transactions on Circuits and Systems I: Regular Papers **52**, p. 13–20 (2005); ISSN 1057-7122; URL http://dx.doi.org/10.1109/tcsi.2004.840093. 10

[93] R. Carmona-Galán, Á. Zarándy, C. Rekeczky, P. Földesy, A. Rodríguez-Pérez, C. Domínguez-Matas, J. Fernández-Berni, G. Liñán-Cembrano, B. Pérez-Verdú, Z. Kárász & et al.; "A hierarchical vision processing architecture oriented to 3D integration of smart camera chips"; Journal of Systems Architecture **59**, p. 908–919 (2013); ISSN 1383-7621; URL http://dx.doi.org/10.1016/j.sysarc.2013.03.002. 10

[94] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. D. Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza & et al.; "A Low Power, Fully Event-Based Gesture Recognition System"; 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017); URL http://dx.doi.org/10.1109/cvpr.2017.781. 10

[95] A. I. Maqueda, A. Loquercio, G. Gallego, N. Garcia & D. Scaramuzza; "Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars"; 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018); URL http://dx.doi.org/10.1109/cvpr.2018.00568. 10

[96] M. B. Milde, O. J. Bertrand, R. Benosmanz, M. Egelhaaf & E. Chicca; "Bioinspired event-driven collision avoidance algorithm based on optic flow"; 2015 International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP) (2015); URL http://dx.doi.org/10.1109/ebccsp.2015.7300673. 10

[97] D. Reverter Valeiras, G. Orchard, S.-H. Ieng & R. B. Benosman; "Neuromorphic Event-Based 3D Pose Estimation"; Frontiers in Neuroscience **9** (2016); ISSN 1662-453X; URL http://dx.doi.org/10.3389/fnins.2015.00522. 10

[98] E. Mueggler, C. Bartolozzi & D. Scaramuzza; "Fast Event-based Corner Detection"; Procedings of the British Machine Vision Conference 2017 (2017); URL http://dx.doi.org/10.5244/c.31.33. 10

[99] A. Lakshmi, A. Chakraborty & C. S. Thakur; "Neuromorphic vision: From sensors to event-based algorithms"; Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery p. e1310 (2019); ISSN 1942-4795; URL http://dx.doi.org/10.1002/widm.1310. 10

[100] G. Orchard, X. Lagorce, C. Posch, S. B. Furber, R. Benosman & F. Galluppi; "Real-time event-driven spiking neural network object recognition on the SpiNNaker platform"; 2015 IEEE International Symposium on Circuits and Systems (ISCAS) (2015); URL http://dx.doi.org/10.1109/iscas.2015.7169171. 10

[101] G. Haessig, A. Cassidy, R. Alvarez, R. Benosman & G. Orchard; "Spiking Optical Flow for Event-Based Sensors Using IBM's TrueNorth Neurosynaptic System"; IEEE Transactions on Biomedical Circuits and Systems **12**, p. 860–870 (2018); ISSN 1940-9990; URL http://dx.doi.org/10.1109/tbcas.2018.2834558. 10

[102] M. Hopkins, G. Pineda-García, P. A. Bogdan & S. B. Furber; "Spiking neural networks for computer vision"; Interface Focus **8**, p. 20180007 (2018); ISSN 2042-8901; URL http://dx.doi.org/10.1098/rsfs.2018.0007. 10

[103] F. Barranco, C. Fermuller & Y. Aloimonos; "Contour Motion Estimation for Asynchronous Event-Driven Cameras"; Proceedings of the IEEE **102**, p. 1537–1556 (2014); ISSN 1558-2256; URL http://dx.doi.org/10.1109/jproc.2014.2347207. 10

[104] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr & T. Delbruck; "Steering a predator robot using a mixed frame/event-driven convolutional neural network"; 2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP) (2016); URL http://dx.doi.org/10.1109/ebccsp.2016.7605233. 10

[105] H. Liu, D. P. Moeys, G. Das, D. Neil, S.-C. Liu & T. Delbruck; "Combined frame- and event-based detection and tracking"; 2016 IEEE International Symposium on Circuits and Systems (ISCAS) (2016); URL http://dx.doi.org/10.1109/iscas.2016.7539103. 10

[106] D. Tedaldi, G. Gallego, E. Mueggler & D. Scaramuzza; "Feature detection and tracking with the dynamic and active-pixel vision sensor (DAVIS)"; 2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP) (2016); URL http://dx.doi.org/10.1109/ebccsp.2016.7605086. 10

[107] G. Indiveri & S.-C. Liu; "Memory and Information Processing in Neuromorphic Systems"; Proceedings of the IEEE **103**, p. 1379–1397 (2015); ISSN 1558-2256; URL http://dx.doi.org/10.1109/jproc.2015.2444094. 10

[108] S. Furber; "Microprocessors: the engines of the digital age"; Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science **473**, p. 20160893 (2017); ISSN 1471-2946; URL http://dx.doi.org/10.1098/rspa.2016.0893. 10

[109] C. L. Lawson, R. J. Hanson, D. R. Kincaid & F. T. Krogh; "Basic Linear Algebra Subprograms for Fortran Usage"; ACM Transactions on Mathematical Software **5**, p. 308–323 (1979); ISSN 0098-3500; URL http://dx.doi.org/10.1145/355841.355847. 12

[110] C. M. Thompson & L. Shure; "Image processing toolbox [for use with Matlab]"; http://infoscience.epfl.ch/record/24814 (1995). 12

[111] G. Bradski; "The OpenCV Library"; Dr. Dobb's Journal of Software Tools (2000). 12

[112] E. Jones, T. Oliphant, P. Peterson *et al.*; "SciPy: Open source scientific tools for Python"; http://www.scipy.org/ (2001). 12

[113] S. Ferg; "Event-Driven Programming: Introduction, Tutorial, History"; http://eventdrivenpgm.sourceforge.net (2006). 12

[114] E. Bainomugisha, A. L. Carreton, T. v. Cutsem, S. Mostinckx & W. d. Meuter; "A survey on reactive programming"; ACM Computing Surveys **45**, p. 1–34 (2013); ISSN 0360-0300; URL http://dx.doi.org/10.1145/2501654.2501666. 12

[115] S. Tilkov & S. Vinoski; "Node.js: Using JavaScript to Build High-Performance Network Programs"; IEEE Internet Computing **14**, p. 80–83 (2010); ISSN 1089-7801; URL http://dx.doi.org/10.1109/mic.2010.145. 12

[116] "Cookie Clicker"; http://orteil.dashnet.org/cookieclicker (2013). 12

[117] "CBS News uses Twitter as part of its news investigating and reporting." https://developer.twitter.com/en/case-studies/cbs-news (2015). 12

[118] "jAER project"; http://jaerproject.org (2007). 13

[119] "cAER library"; https://gitlab.com/inivation/dv-runtime/tags/caer-1.1.2 (2007). 13

[120] "Dynamic Vision System"; https://gitlab.com/inivation/dv-runtime (2019). 13

[121] A. Glover, V. Vasco & C. Bartolozzi; "A Controlled-Delay Event Camera Framework for On-Line Robotics"; 2018 IEEE International Conference on Robotics and Automation (ICRA) (2018); URL http://dx.doi.org/10.1109/icra.2018.8460541. 13

[122] R. Nevatia; "A color edge detector and its use in scene segmentation"; IEEE Transactions on Systems, Man, and Cybernetics **7**, p. 820–826 (1977); ISSN 0018-9472; URL http://dx.doi.org/10.1109/tsmc.1977.4309631. 13

[123] T. Gevers; *Color in Computer Vision: Fundamentals and Applications* (Wiley) (2012); ISBN 0470890843; URL https://www.xarg.org/ref/a/0470890843/. 13

[124] C.-M. Pun & G. Huang; "On-line video object segmentation using illumination-invariant color-texture feature extraction and marker prediction"; Journal of Visual Communication and Image Representation **41**, pp. 391–405 (2016); ISSN 1047-3203; URL http://dx.doi.org/10.1016/j.jvcir.2016.10.017. 13

[125] C. Rother, V. Kolmogorov & A. Blake; ""GrabCut": Interactive Foreground Extraction Using Iterated Graph Cuts"; in "ACM SIGGRAPH 2004 Papers. Los Angeles, CA, USA," SIGGRAPH '04; pp. 309–314 (ACM, New York, NY, USA) (2004); URL http://dx.doi.org/10.1145/1186562.1015720. 13

[126] M. Grundmann, V. Kwatra, M. Han & I. Essa; "Efficient hierarchical graph-based video segmentation"; in "2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. San Francisco, CA, USA," pp. 2141–2148 (2010); ISSN 1063-6919; URL http://dx.doi.org/10.1109/CVPR.2010.5539893. 13

[127] J. Lezama, K. Alahari, J. Sivic & I. Laptev; "Track to the future: Spatio-temporal video segmentation with long-range motion cues"; in "CVPR 2011. Colorado Springs, CO, USA," (IEEE) (2011); ISBN 9781457703942; URL http://dx.doi.org/10.1109/cvpr.2011.6044588. 13

[128] X. Bai, J. Wang & G. Sapiro; "Dynamic Color Flow: A Motion-Adaptive Color Model for Object Segmentation in Video"; in "Computer Vision – ECCV 2010. Heraklion, Crete, Greece," , edited by K. Daniilidis, P. Maragos & N. Paragios; pp. 617–630 (Springer Berlin Heidelberg, Berlin, Heidelberg) (2010); ISBN 978-3-642-15555-0. 13

[129] Y. J. Lee, J. Kim & K. Grauman; "Key-segments for video object segmentation"; in "2011 International Conference on Computer Vision. Barcelona, Spain," pp. 1995–2002 (IEEE) (2011); ISBN 9781457711008; URL http://dx.doi.org/10.1109/iccv.2011.6126471. 13

[130] K. Fukunaga & L. Hostetler; "The estimation of the gradient of a density function, with applications in pattern recognition"; IEEE Transactions on Information

Theory **21**, pp. 32–40 (1975); ISSN 0018-9448; URL http://dx.doi.org/10.1109/tit.1975.1055330. 13

[131] Y. Cheng; "Mean shift, mode seeking, and clustering"; IEEE Transactions on Pattern Analysis and Machine Intelligence **17**, pp. 790–799 (1995); ISSN 0162-8828; URL http://dx.doi.org/10.1109/34.400568. 13

[132] C. Xiao & M. Liu; "Efficient Mean-shift Clustering Using Gaussian KD-Tree"; Computer Graphics Forum **29**, pp. 2065–2073 (2010); ISSN 0167-7055; URL http://dx.doi.org/10.1111/j.1467-8659.2010.01793.x. 14

[133] S. Paris & F. Durand; "A Topological Approach to Hierarchical Segmentation using Mean Shift"; in "2007 IEEE Conference on Computer Vision and Pattern Recognition. Minneapolis, MN, USA," (IEEE) (2007); ISBN 1424411807; URL http://dx.doi.org/10.1109/cvpr.2007.383228. 14

[134] H. Guo, P. Guo & H. Lu; "A Fast Mean Shift Procedure with New Iteration Strategy and Re-sampling"; in "2006 IEEE International Conference on Systems, Man and Cybernetics. Taipei, Taiwan," (IEEE) (2006); ISBN 1424401003; URL http://dx.doi.org/10.1109/icsmc.2006.385220. 14

[135] C. Yang, R. Duraiswami, D. DeMenthon & L. Davis; "Mean-shift analysis using quasiNewton methods"; in "Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429). Barcelona, Spain," , volume 2pp. II–447–50 vol.3 (IEEE) (2003); ISSN 1522-4880; URL http://dx.doi.org/10.1109/icip.2003.1246713. 14

[136] D. Comaniciu; "An algorithm for data-driven bandwidth selection"; IEEE Transactions on Pattern Analysis and Machine Intelligence **25**, pp. 281–288 (2003); ISSN 0162-8828; URL http://dx.doi.org/10.1109/tpami.2003.1177159. 14

[137] D. C. Cireşan, U. Meier, L. M. Gambardella & J. Schmidhuber; "Deep, Big, Simple Neural Nets for Handwritten Digit Recognition"; Neural Computation **22**, p. 3207–3220 (2010); ISSN 1530-888X; URL http://dx.doi.org/10.1162/neco_a_00052. 14

[138] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard & L. D. Jackel; "Backpropagation Applied to Handwritten Zip Code Recognition"; Neural Computation **1**, p. 541–551 (1989); ISSN 1530-888X; URL http://dx.doi.org/10.1162/neco.1989.1.4.541. 14

[139] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke & A. Rabinovich; "Going Deeper with Convolutions"; in "Computer Vision and Pattern Recognition (CVPR)," (2015); URL http://arxiv.org/abs/1409.4842. 14

[140] K. He, X. Zhang, S. Ren & J. Sun; "Deep Residual Learning for Image Recognition"; 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016); URL http://dx.doi.org/10.1109/cvpr.2016.90. 14

[141] A. Flachot & K. R. Gegenfurtner; "Processing of chromatic information in a deep convolutional neural network"; Journal of the Optical Society of America A **35**, p. B334 (2018); ISSN 1520-8532; URL http://dx.doi.org/10.1364/josaa.35.00b334. 14, 41

[142] A. Krizhevsky, I. Sutskever & G. E. Hinton; "ImageNet classification with deep convolutional neural networks"; Communications of the ACM **60**, p. 84–90 (2017); ISSN 0001-0782; URL http://dx.doi.org/10.1145/3065386. 14, 15

[143] F. L. Roubieu, J. R. Serres, F. Colonnier, N. Franceschini, S. Viollet & F. Ruffier; "A biomimetic vision-based hovercraft accounts for bees' complex behaviour in various corridors"; Bioinspiration & Biomimetics **9**, p. 036003 (2014); ISSN 1748-3190; URL http://dx.doi.org/10.1088/1748-3182/9/3/036003. 15

[144] D. P. Moeys, F. Corradi, C. Li, S. A. Bamford, L. Longinotti, F. F. Voigt, S. Berry, G. Taverni, F. Helmchen & T. Delbruck; "A Sensitive Dynamic and Active Pixel Vision Sensor for Color or Neural Imaging Applications"; IEEE Transactions on Biomedical Circuits and Systems **12**, p. 123–136 (2018); ISSN 1940-9990; URL http://dx.doi.org/10.1109/tbcas.2017.2759783. 15

[145] F. D. Broccard, S. Joshi, J. Wang & G. Cauwenberghs; "Neuromorphic neural interfaces: from neurophysiological inspiration to biohybrid coupling with nervous systems"; Journal of Neural Engineering **14**, p. 041002 (2017); ISSN 1741-2552; URL http://dx.doi.org/10.1088/1741-2552/aa67a9. 16

[146] J. Tapson, J. Diaz, D. Sander, N. Gurari, E. Chicca, P. Pouliquen & R. Etienne-Cummings; "The feeling of color: A haptic feedback device for the visually disabled"; 2008 IEEE Biomedical Circuits and Systems Conference (2008); URL http://dx.doi.org/10.1109/biocas.2008.4696954. 16

[147] N. Gaspar, A. Sondhi, B. Evans & K. Nikolic; "A low-power neuromorphic system for retinal implants and sensory substitution"; 2016 IEEE Biomedical Circuits and Systems Conference (BioCAS) (2016); URL http://dx.doi.org/10.1109/biocas.2016.7833729. 16

[148] F. Galluppi, D. Pruneau, J. Chavas, X. Lagorce, C. Posch, G. Chenegros, G. Cordurie, C. Galle, N. Oddo & R. Benosman; "A stimulation platform for optogenetic and bionic vision restoration"; 2017 IEEE International Symposium on Circuits and Systems (ISCAS) (2017); URL http://dx.doi.org/10.1109/iscas.2017.8050683. 16

[149] T. A. Weissman, J. R. Sanes, J. W. Lichtman & J. Livet; "Generating and Imaging Multicolor Brainbow Mice"; Cold Spring Harbor Protocols **2011**, p. pdb.top114–pdb.top114 (2011); ISSN 1559-6095; URL http://dx.doi.org/10.1101/pdb.top114. 16

[150] S. Barrett; "stb GitHub page"; https://github.com/nothings/stb (2014). 19

[151] T. Veldhuizen; "Techniques for scientific C++"; Computer science technical report **542**, p. 60 (2000). 20

[152] H. Blum, A. Dietmüller, M. Milde, J. Conradt, G. Indiveri & Y. Sandamirskaya; "A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor"; Robotics: Science and Systems XIII (2017); URL http://dx.doi.org/10.15607/rss.2017.xiii.035. 24

[153] A. Appleby; "SMHasher"; https://github.com/aappleby/smhasher (2014). 25

[154] G. K. Cohen; *Event-Based Feature Detection, Recognition and Classification*; Ph.D. thesis; Université Pierre et Marie Curie - Paris VI, Western Sydney University (2016). 29

[155] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi & R. B. Benosman; "HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition"; IEEE Transactions on Pattern Analysis and Machine Intelligence **39**, p. 1346–1359 (2017); ISSN 2160-9292; URL http://dx.doi.org/10.1109/tpami.2016.2574707. 29

[156] "Qt"; https://www.qt.io (1995). 32

[157] U. A. Acar, V. Aksenov, A. Charguéraud & M. Rainey; "Performance challenges in modular parallel programs"; Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming - PPoPP '18 (2018); URL http://dx.doi.org/10.1145/3178487.3178516. 37

[158] V. Tovinkere & M. Voss; "Flow Graph Designer: A Tool for Designing and Analyzing Intel Threading Building Blocks Flow Graphs"; 2014 43rd International Conference on Parallel Processing Workshops (2014); URL http://dx.doi.org/10.1109/icppw.2014.31. 37

[159] P. Rogister, R. Benosman, S.-H. Ieng, P. Lichtsteiner & T. Delbruck; "Asynchronous Event-Based Binocular Stereo Matching"; IEEE Transactions on Neural Networks and Learning Systems **23**, pp. 347–353 (2012); ISSN 2162-2388; URL http://dx.doi.org/10.1109/tnnls.2011.2180025. 41

[160] J. Carneiro, S.-H. Ieng, C. Posch & R. Benosman; "Event-based 3D reconstruction from neuromorphic retinas"; Neural Networks **45**, p. 27–38 (2013); ISSN 0893-6080; URL http://dx.doi.org/10.1016/j.neunet.2013.03.006. 41

[161] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng & C. Bartolozzi; "Event-Based Visual Flow"; IEEE Transactions on Neural Networks and Learning Systems **25**, pp. 407–417 (2014); ISSN 2162-2388; URL http://dx.doi.org/10.1109/tnnls.2013.2273537. 41

[162] Z. Ni, A. Bolopion, J. Agnus, R. Benosman & S. Regnier; "Asynchronous Event-Based Visual Shape Tracking for Stable Haptic Feedback in Microrobotics"; IEEE Transactions on Robotics **28**, pp. 1081–1089 (2012); ISSN 1941-0468; URL http://dx.doi.org/10.1109/tro.2012.2198930. 41

[163] C. Posch, D. Matolin & R. Wohlgenannt; "A QVGA 143 dB Dynamic Range Frame-Free PWM Image Sensor With Lossless Pixel-Level Video Compression and Time-Domain CDS"; IEEE Journal of Solid-State Circuits **46**, pp. 259–275 (2011); ISSN 1558-173X; URL http://dx.doi.org/10.1109/jssc.2010.2085952. 41, 54

[164] D. Drazen, P. Lichtsteiner, P. Hafliger, T. Delbruck & A. Jensen; "Toward real-time particle tracking using an event-based dynamic vision sensor"; Experiments in Fluids **51**, pp. 1465–1469 (2011); ISSN 1432-1114; URL http://dx.doi.org/10.1007/s00348-011-1207-y. 41

[165] D. Reverter Valeiras, X. Lagorce, X. Clady, C. Bartolozzi, S.-H. Ieng & R. Benosman; "An Asynchronous Neuromorphic Event-Driven Visual Part-Based Shape Tracking"; IEEE Transactions on Neural Networks and Learning Systems **26**, p. 3045–3059 (2015); ISSN 2162-2388; URL http://dx.doi.org/10.1109/tnnls.2015.2401834. 42

[166] R. Hartley; "In defense of the eight-point algorithm"; IEEE Transactions on Pattern Analysis and Machine Intelligence **19**, pp. 580–593 (1997); ISSN 0162-8828; URL http://dx.doi.org/10.1109/34.601246. 42

[167] A. Geiger, F. Moosmann, O. Car & B. Schuster; "Automatic camera and range sensor calibration using a single shot"; in "2012 IEEE International Conference on Robotics and Automation. St Paul, MN, USA.", (IEEE) (2012); URL http://dx.doi.org/10.1109/icra.2012.6224570. 42

[168] H. Cheng, X. Jiang, Y. Sun & J. Wang; "Color image segmentation: advances and prospects"; Pattern Recognition **34**, pp. 2259–2281 (2001); ISSN 0031-3203; URL http://dx.doi.org/10.1016/s0031-3203(00)00149-7. 44

[169] A. K. Jain; *Fundamentals of Digital Image Processing (Prentice Hall Information and System Sciences Series)* (Pearson) (1988); ISBN 0133361659; URL https://www.amazon.com/Fundamentals-Digital-Image-Processing-Anil/dp/0133361659. 45

[170] J. C. Lagarias, J. A. Reeds, M. H. Wright & P. E. Wright; "Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions"; SIAM Journal on Optimization **9**, pp. 112–147 (1998); URL http://dx.doi.org/10.1137/s1052623496303470. 45

[171] S. Suzuki & K. be; "Topological structural analysis of digitized binary images by border following"; Computer Vision, Graphics, and Image Processing **30**, pp. 32–46 (1985); ISSN 0734-189X; URL http://dx.doi.org/10.1016/0734-189x(85)90016-7. 49

[172] D. P. Moeys, C. Li, J. N. Martel, S. Bamford, L. Longinotti, V. Motsnyi, D. S. S. Bello & T. Delbruck; "Color temporal contrast sensitivity in dynamic vision sensors"; 2017 IEEE International Symposium on Circuits and Systems (ISCAS) (2017); URL http://dx.doi.org/10.1109/iscas.2017.8050412. 51, 82

[173] T. Hansen & K. R. Gegenfurtner; "Color contributes to object-contour perception in natural scenes"; Journal of Vision **17**, p. 14 (2017); ISSN 1534-7362; URL http://dx.doi.org/10.1167/17.3.14. 54

[174] J. Davis, Y.-H. Hsieh & H.-C. Lee; "Humans perceive flicker artifacts at 500 Hz"; Scientific Reports **5** (2015); ISSN 2045-2322; URL http://dx.doi.org/10.1038/srep07861. 57, 100

[175] T. Gollisch & M. Meister; "Eye Smarter than Scientists Believed: Neural Computations in Circuits of the Retina"; Neuron **65**, p. 150–164 (2010); ISSN 0896-6273; URL http://dx.doi.org/10.1016/j.neuron.2009.12.009. 57

[176] S. Martinez-Conde, S. L. Macknik, X. G. Troncoso & D. H. Hubel; "Microsaccades: a neurophysiological analysis"; Trends in Neurosciences **32**, p. 463–475 (2009); ISSN 0166-2236; URL http://dx.doi.org/10.1016/j.tins.2009.05.006. 57

[177] "EventIDE"; http://www.okazolab.com (2012). 57

[178] L. Sawides, A. Gambín-Regadera, A. de Castro & P. Artal; "High speed visual stimuli generator to estimate the minimum presentation time required for an orientation discrimination task"; Biomedical Optics Express **9**, p. 2640 (2018); ISSN 2156-7085; URL http://dx.doi.org/10.1364/boe.9.002640. 58

[179] C. H. Poth, R. M. Foerster, C. Behler, U. Schwanecke, W. X. Schneider & M. Botsch; "Ultrahigh temporal resolution of visual presentation using gaming monitors and G-Sync"; Behavior Research Methods **50**, p. 26–38 (2018); ISSN 1554-3528; URL http://dx.doi.org/10.3758/s13428-017-1003-6. 58

[180] S. Kime, F. Galluppi, X. Lagorce, R. B. Benosman & J. Lorenceau; "Psychophysical Assessment of Perceptual Performance With Varying Display Frame Rates"; Journal of Display Technology **12**, p. 1372–1382 (2016); ISSN 1558-9323; URL http://dx.doi.org/10.1109/jdt.2016.2603222. 58

[181] M. A.Khoei, F. Galluppi, Q. Sabatier, P. Pouget, B. R Cottereau & R. Benosman; "Faster is better: Visual responses to motion are stronger for higher refresh rates"; bioRxiv (2018); URL http://dx.doi.org/10.1101/505354. 58

[182] L. Abeni, A. Goel, C. Krasic, J. Snow & J. Walpole; "A measurement-based analysis of the real-time performance of linux"; Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (2002); URL http://dx.doi.org/10.1109/rttas.2002.1137388. 60

[183] V. Deep & T. Elarabi; "HEVC/H.265 vs. VP9 state-of-the-art video coding comparison for HD and UHD applications"; 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE) (2017); URL http://dx.doi.org/10.1109/ccece.2017.7946796. 63

[184] F. Sivrikaya & B. Yener; "Time synchronization in sensor networks: a survey"; IEEE Network **18**, p. 45–50 (2004); ISSN 0890-8044; URL http://dx.doi.org/10.1109/mnet.2004.1316761. 66

[185] "FFmpeg library"; https://ffmpeg.org (2000). 70

[186] "gstreamer library"; https://gstreamer.freedesktop.org (2001). 70

[187] "libav library"; https://www.libav.org (2011). 70

[188] "GLFW library"; https://www.glfw.org (2002). 72

[189] T. S. Weber, M. Dukes, D. C. Miles, S. P. Glaser, S. H. Naik & K. R. Duffy; "Site-specific recombinatorics: in situ cellular barcoding with the Cre Lox system"; BMC Systems Biology **10** (2016); ISSN 1752-0509; URL http://dx.doi.org/10.1186/s12918-016-0290-3. 81, 82

[190] X. Berthelon, G. Chenegros, T. Finateu, S.-H. Ieng & R. Benosman; "Effects of Cooling on the SNR and Contrast Detection of a Low-Light Event-Based Camera"; IEEE Transactions on Biomedical Circuits and Systems **12**, p. 1467–1474 (2018); ISSN 1940-9990; URL http://dx.doi.org/10.1109/tbcas.2018.2875202. 82

[191] W. Pei, T. B. Feyerabend, J. Rössler, X. Wang, D. Postrach, K. Busch, I. Rode, K. Klapproth, N. Dietlein, C. Quedenau & et al.; "Polylox barcoding reveals haematopoietic stem cell fates realized in vivo"; Nature **548**, p. 456–460 (2017); ISSN 1476-4687; URL http://dx.doi.org/10.1038/nature23653. 82, 92, 94

[192] L. V. Bystrykh & M. E. Belderbos; "Clonal Analysis of Cells with Cellular Barcoding: When Numbers and Sizes Matter"; Stem Cell Heterogeneity p. 57–89 (2016); ISSN 1940-6029; URL http://dx.doi.org/10.1007/7651_2016_343. 82

[193] Y. Wei & A. A. Koulakov; "An Exactly Solvable Model of Random Site-Specific Recombinations"; Bulletin of Mathematical Biology **74**, p. 2897–2916 (2012); ISSN 1522-9602; URL http://dx.doi.org/10.1007/s11538-012-9788-z. 82

[194] D. Cai, K. B. Cohen, T. Luo, J. W. Lichtman & J. R. Sanes; "Improved tools for the Brainbow toolbox"; Nature Methods **10**, p. 540–547 (2013); ISSN 1548-7105; URL http://dx.doi.org/10.1038/nmeth.2450. 82

[195] T. Höfer, M. Barile, K. Busch, T. Feyerabend, W. Pei, J. Rössler, A.-K. Schuon, X. Wang & H.-R. Rodewald; "Quantitating native hematopoiesis"; Experimental Hematology **53**, p. S26 (2017); ISSN 0301-472X; URL http://dx.doi.org/10.1016/j.exphem.2017.06.014. 82

[196] A. Reibman & K. Trivedi; "Numerical transient analysis of markov models"; Computers & Operations Research **15**, p. 19–36 (1988); ISSN 0305-0548; URL http://dx.doi.org/10.1016/0305-0548(88)90026-3. 86

[197] M. M. Shahshahani; "Statistics 217/218: Introduction to Stochastic Processes, chapter 3"; https://web.stanford.edu/class/stat217/Chapter3.pdf (2003). 86

[198] V. G. Kulkarni; *Modeling and Analysis of Stochastic Systems (Chapman & Hall/CRC Texts in Statistical Science)* (Chapman and Hall/CRC) (2016); ISBN 9781498756617; URL https://www.xarg.org/ref/a/1498756611/. 86

[199] M. Kijima; *Markov Processes for Stochastic Modeling (Stochastic Modeling Series)* (Chapman and Hall/CRC) (1997); ISBN 0412606607; URL https://www.xarg.org/ref/a/0412606607/. 88

[200] J. G. Kemeny; *Finite Markov Chains: With a New Appendix "Generalization of a Fundamental Matrix" (Undergraduate Texts in Mathematics)* (Springer) (1983); ISBN 0387901922; URL https://www.xarg.org/ref/a/0387901922/. 89

[201] R. Hoess, A. Wierzbicki & K. Abremski; "Formation of small circular DNA molecules via an in vitro site-specific recombination system"; Gene **40**, p. 325–329 (1985); ISSN 0378-1119; URL http://dx.doi.org/10.1016/0378-1119(85)90056-3. 90

[202] B. Zheng, M. Sage, E. A. Sheppeard, V. Jurecic & A. Bradley; "Engineering Mouse Chromosomes with Cre-loxP: Range, Efficiency, and Somatic Applications"; Molecular and Cellular Biology **20**, p. 648–655 (2000); ISSN 0270-7306; URL http://dx.doi.org/10.1128/mcb.20.2.648-655.2000. 90

[203] Z. Caner Taskin; "Tutorial Guide to Mixed-Integer Programming Models and Solution Techniques"; Optimization in Medicine and Biology (2008); ISSN 2155-854X; URL http://dx.doi.org/10.1201/9780849305696.axa. 92

[204] K. Loulier, R. Barry, P. Mahou, Y. Le Franc, W. Supatto, K. Matho, S. Ieng, S. Fouquet, E. Dupin, R. Benosman & et al.; "Multiplex Cell and Lineage Tracking with Combinatorial Labels"; Neuron **81**, p. 505–520 (2014); ISSN 0896-6273; URL http://dx.doi.org/10.1016/j.neuron.2013.12.016. 92

[205] R. Rajaram, B. Castellani & A. N. Wilson; "Advancing Shannon Entropy for Measuring Diversity in Systems"; Complexity **2017**, p. 1–10 (2017); ISSN 1099-0526; URL http://dx.doi.org/10.1155/2017/8715605. 92

[206] T. Saraf-Levy, S. W. Santoro, H. Volpin, T. Kushnirsky, Y. Eyal, P. G. Schultz, D. Gidoni & N. Carmi; "Site-specific recombination of asymmetric lox sites mediated by a heterotetrameric Cre recombinase complex"; Bioorganic & Medicinal Chemistry **14**, p. 3081–3089 (2006); ISSN 0968-0896; URL http://dx.doi.org/10.1016/j.bmc.2005.12.016. 96

[207] E. Jonas & K. P. Kording; "Could a Neuroscientist Understand a Microprocessor?" PLOS Computational Biology **13**, p. e1005268 (2017); ISSN 1553-7358; URL http://dx.doi.org/10.1371/journal.pcbi.1005268. 100

[208] M. Elowitz & W. A. Lim; "Build life to understand it"; Nature **468**, p. 889–890 (2010); ISSN 1476-4687; URL http://dx.doi.org/10.1038/468889a. 100

[209] R. Douglas, M. Mahowald & K. Martin; "Neuroinformatics as Explanatory Neuroscience"; NeuroImage **4**, p. S25–S28 (1996); ISSN 1053-8119; URL http://dx.doi.org/10.1006/nimg.1996.0046. 100

[210] M. Alioto, V. De & A. Marongiu; "Energy-Quality Scalable Integrated Circuits and Systems: Continuing Energy Scaling in the Twilight of Moore's Law";

IEEE Journal on Emerging and Selected Topics in Circuits and Systems **8**, p. 653–678 (2018); ISSN 2156-3365; URL http://dx.doi.org/10.1109/jetcas.2018.2881461. 101

[211] S. Reiss; "Carver Mead's natural inspiration"; Technology Review -Manchester Nh- **107**, pp. 76–80 (2004). 101

[212] A. Brychtová & A. Çöltekin; "The effect of spatial distance on the discriminability of colors in maps"; Cartography and Geographic Information Science **44**, p. 229–245 (2016); ISSN 1545-0465; URL http://dx.doi.org/10.1080/15230406.2016.1140074. 102

[213] J. A. Leñero-Bardallo, R. Carmona-Galán & A. Rodríguez-Vázquez; "Applications of event-based image sensors-Review and analysis"; International Journal of Circuit Theory and Applications (2018); ISSN 0098-9886; URL http://dx.doi.org/10.1002/cta.2546. 102

[214] P. Martínez-Cañada, C. Morillas, B. Pino, E. Ros & F. Pelayo; "A Computational Framework for Realistic Retina Modeling"; International Journal of Neural Systems **26**, p. 1650030 (2016); ISSN 1793-6462; URL http://dx.doi.org/10.1142/s0129065716500301. 103

[215] H. Wei & X. Guan; "The Simulation of Early Vision in Biological Retina and Analysis of Its Performance"; 2008 Congress on Image and Signal Processing (2008); URL http://dx.doi.org/10.1109/cisp.2008.544. 103

[216] P. Abshire, A. Bermak, R. Berner, G. Cauwenberghs, S. Chen, J. B. Christen, T. Constandinou, E. Culurciello, M. Dandin, T. Datta & et al.; "Confession session: Learning from others mistakes"; 2011 IEEE International Symposium of Circuits and Systems (ISCAS) (2011); URL http://dx.doi.org/10.1109/iscas.2011.5937774. 103

[217] H. Rebecq, D. Gehrig & D. Scaramuzza; "ESIM: an Open Event Camera Simulator"; in "Proceedings of The 2nd Conference on Robot Learning," , *Proceedings of Machine Learning Research*, volume 87, edited by A. Billard, A. Dragan, J. Peters & J. Morimoto; pp. 969–982 (PMLR) (2018); URL http://proceedings.mlr.press/v87/rebecq18a.html. 103

[218] M. L. Katz, K. Nikolic & T. Delbruck; "Live demonstration: Behavioural emulation of event-based vision sensors"; 2012 IEEE International Symposium on Circuits and Systems (2012); URL http://dx.doi.org/10.1109/iscas.2012.6272143. 103

[219] D. Grois, D. Marpe, A. Mulayoff, B. Itzhaky & O. Hadar; "Performance comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC encoders"; 2013 Picture Coding Symposium (PCS) (2013); URL http://dx.doi.org/10.1109/pcs.2013.6737766. 105