An Event-by-Event Feature Detection and Tracking Invariant to Motion Direction and Velocity

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

15-11-2021 / 19-11-2021

CITATION

Dardelet, Laurent; Benosman, Ryad; Ieng, Sio-Hoi (2021): An Event-by-Event Feature Detection and Tracking Invariant to Motion Direction and Velocity. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.17013824.v1>

DOI

[10.36227/techrxiv.17013824.v1](https://doi.org/10.36227/techrxiv.17013824.v1)

An Event-by-Event Feature Detection and Tracking Invariant to Motion Direction and Velocity

Laurent Dardet¹, Sio-Hoi Ieng¹, Ryad Benosman^{1,2,3}

ABSTRACT

This paper presents a new event-based method for selecting and tracking features from the output of an event-based camera while being robust to motion direction and velocity changes. It solves for the first time the current problem of establishing event correspondences across time without requiring a minimization on time slices or frames of accumulated events. Unlike existing tracking algorithms, the method is designed to detect generic features and does not need to operate on particular predefined shapes such as corners. It relies on a new framework for event-based computation that detects finite curvature structures by estimating instantaneously, both the speed and position from local flow estimations. Results on several real environments are shown taking into account large variations in illuminance, speed, and type of features. The combined use of both the velocity and spatial information allows for robustness against occlusion. The method is parameters-tuning free as it dynamically adapts to the state of trackers.

1 INTRODUCTION

This paper introduces an event-based solution for selecting and tracking features from the output of an event-based camera. It overcomes for the first time the hard issue of extracting stable features from temporal information output by the sensor while being independent from motion direction and velocity changes. This is a major issue when using these sensors as the acquired space-time features can vary drastically although the underlying spatial structure is the same. This paper is based on a pure event per event methodology that allows it to make the most of event-based cameras' native properties of low latency and power. It permits for the first time to formulate the problem of features' selection and tracking in the time domain as an inter-dependent process between detection, tracking and continuous velocity

estimation. The method does not assume any priors of what a feature is.

Event based acquisition introduces a major shift in the way visual information is being processed. Unlike conventional cameras that produce absolute intensity images at a fixed rate, event cameras rely on a data driven acquisition process where independent pixels asynchronously signal light intensity changes (coined as "events") at a high temporal resolution (usually around $1\mu s$).

The use of sparse information represented by visual events can be puzzling to a conventional computer vision scientist used to deal with snapshots, gray levels and colors. Processing visual events requires new thinking to derive a novel family of algorithms capable of operating efficiently in the time domain by reducing memory footprints and computational power requirements.

With continuously improved spatial resolutions, almost matching conventional sensors (from VGA to Mega Pixel), the event-based vision sensors' high temporal resolution allows the measurement of velocities in the focal plane with great precision. This is the most valuable and reliable source of information of these sensors. It has been extensively used in this paper.

Presently, there are two opposite philosophies in processing data acquired by event cameras. The first one is advocating for the use of optimization techniques on static images of accumulated events or batches of events defined over different durations and sizes. This approach only uses the high dynamic range properties of these sensors, the recurring motivation behind this is to reuse decades of conventional computer vision techniques [1][2] and avoid using single events that are wrongly considered to be too noisy and carrying little information. This approach results in large wastes of resources and an increased latency. The second approach is closer to the event-based acquisition properties. It is advocating for a local space-time processing of individual events, as they occur, with the idea that each one of them is contributing by an infinitesimal information to the sensing problem to solve. This approach naturally saves power and memory footprint because it only stores what is required and computes using the minimal amount of resources only when something has changed in the scene.

Previous works that tackled feature detection from the output of an event-based sensor include several corners detectors that

¹INSERM UMR I S 968; Sorbonne Universites, UPMC Univ Paris 06, UMR S 968; CNRS, UMR 7210, Institut de la Vision, F-75012, Paris, France

² University of Pittsburgh , Biomedical Science Tower 3, Fifth Avenue, Pittsburgh, PA, USA

³ Carnegie Mellon University , Robotics Institute, 5000 Forbes Avenue Pittsburgh PA 15213-3890, USA

are inferred from the optical flow computation [3], or obtained by using Harris operator on locally integrated temporal frame of events [4]. The time-surface as introduced in [5] is another way to extract dynamic patterns (which also includes corners) as it allows to define a compact feature-velocity descriptor around an incoming event. In [6], the time-surface allows to build a feature that can then be used with FAST [2] detector. In [7], a FAST detector is applied to the time surface to find corners and a graph-based technique is jointly used to find the closest corner in space and time that support motions assumption to achieve tracking. An improvement to this approach is introduced in [8] with the idea of using a local descriptor built from the time surface. The use of descriptors allows for tracking via minimizing descriptor differences. A hybrid class of approaches combine frames and events to overcome the difficulty of pattern selection and tracking using events as the same scene pattern can produce different events depending on the motion direction, making the event correspondences across time very challenging. In [9], the DAVIS sensor is used at the rate of 35 fps to extract corners and other traditional image features are detected on frames by applying standard computer vision detectors, while the tracking is operated on the events captured between the frames [10]. The same idea can be also found in [11]. An expectation-maximization approach, coupled with an iterative closest point algorithm is used in [12] to keep track of corners that are detected at the beginning of the sequences from an edge map built by accumulating events over a manually selected integration time. Events are integrated over a duration deduced from the optical flow and the edge are mapped to generate unitary contours to then apply an ICP algorithm [13]. The approach proposed in [14], based on global optimization of a cost function over a cluster of events, aims to find the motion parameters that yield a homography that maps the cluster of events into an “event image” with a maximal contrast. This approach while not based on features, still deals with the data association problem. It has some overlapping properties with our approach, such as the ability to relate the sensor’s motion to the data, however they differ deeply due to their synchronous/asynchronous nature.

2 EVENT-BASED SENSORS

Event-based vision sensors are gaining popularity within the computer vision community as they offer many advantages over conventional frame-based cameras that cannot be matched without an unreasonable increase in computational resources. Low computation requirements are achieved by reducing redundant visual data at the level of pixels. This naturally allows lower latency and precise temporal acquisition. Event-based cameras (Figure 1(a)) are based on an asynchronous level crossing sampling as shown in Figure 1(b)-(c).

Most of the available event-based vision sensors stem from the Dynamic Vision Sensor (DVS) [16]. As such, they work in a similar manner by capturing relative illuminance changes: each individual pixel emits an “event” each time the illuminance crosses a predefined threshold at μs precision. An event contains both the pixel coordinates from which it originates and the time it is triggered, hence it is usually defined as $e = [x, t]$.

Figure 1: Event-based acquisition of visual signals: (a) the ATIS Event-based camera used in this paper [15], (b) principle of event-based sampling. The variations of the logarithm of the light intensity of a pixel located at $[x, y]^T$ over time, (c) asynchronous temporal contrast events generated each time the light intensity crosses a level in $\log(I)$, (d) the output of a conventional frame based cameras at some 30-60Hz vs in (e) the high temporal precision event-based output of the sensor at around (1 μs).

Additional information can be added such as the sign of that change as shown in Figure 1(c) and conventionally referred to as “ON/OFF” polarities. Other variations of the silicon retinas implement functions such as capturing absolute illuminance information with the same asynchronous principle as shown in Figure 1(e) [15][17] (not used in this work) or by implementing a less advantageous hybrid solution that also acquires conventional images [9]. The asynchronous level crossing sampling allows these sensors to reach over 120dB in dynamic range.

The presented work is carried out using a 640×480 pixels Asynchronous Time-based Image Sensor (ATIS) [18] and the Event-Camera Dataset [19] captured with a DAVIS240C [20].

3 INCREMENTAL CONTINUOUS DETECTION AND TRACKING

3.1 Trackers definition, position and angle update

We define a tracker T as a set $\{\mathbf{X}(t), \mathbf{v}(t), \theta(t), \omega(t)\}$, where $\mathbf{X}(t)$ is its position in the focal plane, $\mathbf{v}(t)$ its translation velocity, $\theta(t)$ its rotation angle w.r.t the optical axis z and $\omega(t)$ its rotation speed. An event output by the neuromorphic camera indexed by n is defined as the pair $e_n = \{\mathbf{X}_n, t_n\}$, with $\mathbf{X}_n$ its 2D spatial coordinates in the focal plane and t_n its time of occurrence. An incoming event is “assigned” to a tracker T when it is close enough to the tracker i.e. satisfying the spatial constraint: $\|\mathbf{X}_n - \mathbf{X}\| \leq R$, where R is the spatial radius of the tracker (as shown in Figure 2).

If we assume a constant velocity motion of the tracker between consecutive event - a reasonable assumption due to the low

Figure 2: A tracker T located at $\mathbf{X}$ is assigned an incoming event e_n at location $\mathbf{X}_n$ if $\|\mathbf{X}_n - \mathbf{X}\| \leq R$, both expressed in $\mathcal{R}_f$ the coordinate reference of the focal plane

latency and the high temporal accuracy of the event-based sensors - we can update its position $\mathbf{X}(t_n)$ and angle $\theta(t_n)$ at the time of the arrival of e_n as:

$$\begin{cases} \mathbf{X}(t_n) = \mathbf{X}(t_{n-1}) + (t_n - t_{n-1})\mathbf{v}(t_{n-1}) \\ \theta(t_n) = \theta(t_{n-1}) + (t_n - t_{n-1})\omega(t_{n-1}) \end{cases} \quad (1)$$

This assumes that $\mathbf{v}$ and ω can be estimated at any time. This is partially achieved by computing the optical flow as explained later.

We define the location of that event in the tracker coordinate frame as $\mathbf{x}_n$, such that

$$\mathbf{x}_n = \varrho(-\theta(t_n))(\mathbf{X}_n - \mathbf{X}(t_n)) \quad (2)$$

with $\varrho(\theta)$ the 2D rotation matrix of angle θ .

3.2 Tracker activity

To enable the continuous update of a tracker T , we first define the time decay factor λ_n associated with T as:

$$\lambda_n = \exp(-(t_n - t_{n-1})/\tau(t_{n-1})). \quad (3)$$

where $\tau(t_n)$ is the time needed for the tracker to move by one pixel. The estimation of this dynamic time property of the tracker will be fully expressed in section 3.6. Times t_n and t_{n-1} correspond respectively to time of arrival of the incoming event e_n and the preceding one e_{n-1} .

We then define $\mathcal{A}$, initialized to 0 at t_0 , as a measure of the activity of a tracker T :

$$\mathcal{A}(t_n) = \mathcal{A}(t_{n-1})\lambda_n + 1. \quad (4)$$

$\mathcal{A}$ is updated by each incoming event that is assigned to a tracker T . This form is thus a sliding average over a time period τ . Each variable defined by a similar equation is thus robust to noise, as well as reactive to variations on a timescale τ .

3.3 Optical flow

For an incoming event $e_n = \{\mathbf{x}_n, t_n\}$ assigned to T , we define the set $\mathcal{E}_n$ that contains all events that occurred in the time interval $[t_n - N_\tau(t_n), t_n]$, within a distance r to $\mathbf{x}_n$ (shown as a black circle in Figure 3):

$$\begin{aligned} \mathcal{E}_n = \{e_k = \{\mathbf{x}_k, t_k\} \mid \|\mathbf{x}_k - \mathbf{x}_n\| \leq r \\ \text{and } t_k \in [t_n - N_\tau(t_n), t_n]\}. \end{aligned} \quad (5)$$

The quantity $N_\tau(t) = N \cdot \tau$ with τ the time needed for a unitary contour to move by one pixel, and N the number of lines considered. $N_\tau(t)$ expresses the time needed by the tracker to move by N pixels, it also defines an adaptive time window parametrized by the velocity of the tracker.

With $\mathcal{E}_n$, we estimate the local optical flow $\mathbf{f}_n$ (shown in blue in Figure 3) at $\mathbf{x}_n$ at time t_n as:

$$\mathbf{f}_n = (\mathbf{x}_n - \bar{\mathbf{x}}_k)/(t_n - \bar{t}_k), \quad (6)$$

where $\bar{\mathbf{x}}_k$ and $\bar{t}_k$ are the spatial position and the timestamp averaged over the set $\mathcal{E}_n$.

Figure 3: The spatial coordinates of an incoming event e_n are now expressed in the tracker T coordinates system $\mathcal{R}_T$ at location $\mathbf{x}_n$. The optical flow $\mathbf{f}_n$ is computed as the difference between $\mathbf{x}_n$ and the mean positions of previous events happening in the temporal window $[t_n - N_\tau(t_n), t_n]$ contained in the set $\mathcal{E}_n$

Remark1:

The computation of the optical flow as a difference between the mean spatio-temporal position of $\mathcal{E}_n$ and the current event works reliably provided one considers small spatial neighborhoods r (usually set between 2-4 pixels). The reliable direction of the estimate flow is the one orthogonal to local edges due to ambiguities related to the aperture problem. We will show in the following sections that local average flow allows the tracker to track local edges by introducing a metric that quantifies the aperture problem. We intentionally choose this simple form of flow instead of more accurate incremental flow computations techniques such as the canonical local plane fitting regularization used in [5][21]. The main reason being it is computationally less prohibitive than most of the existing event-based optical flow algorithms and still proves to converge to an accurate flow estimation with the continuous update of the tracking procedure.

Remark2:

In the tracker's coordinate frame, all events generated by the same edge will follow a narrow spatial distribution that outlines the edge once the tracker velocity converges to the structure's one as shown in [22], [23]. The estimated flow, in the tracker's

coordinate frame being the relative velocity between the tracker and the structure, will converge to $\mathbf{0}$ when the estimation is correct [24].

3.4 Contour velocity estimation

A 3 degree of freedom (2 for the translation and 1 for the rotation) tracker cannot be properly estimated from the optical flow alone as the flow provides only two translational components. The rotational component can be calculated from an estimator that integrates the optical flow over the time period τ , based on the motion equation expressed in the tracker reference frame. We define $\mathbf{v}_c = (v_{cx}, v_{cy})^T$ and ω_c respectively the translation and rotation velocities of the contour in $\mathcal{R}_T$. These are the velocities we want to estimate and nullify by correcting the tracker velocities $\mathbf{v}$ and ω .

For any incoming event $e_n = \{\mathbf{x}_n, t_n\}$, and defining $\mathbf{x}_n = (x_n, y_n)^T$, the instantaneous velocity on the object at the event location is

$$\mathbf{v}_n = \mathbf{v}_c + \omega_c \mathbf{o}_n, \quad (7)$$

where $\mathbf{o}_n = (-y_n, x_n)^T$, the resulting vector in the focal plane of the cross product of $\mathbf{z} = (0, 0, 1)^T$ and $(\mathbf{x}_n^T, 0)^T$.

We then define the unit vector normal to that edge as $\mathbf{u}_n = (c_n, s_n)^T$. Hence, the optical flow $\mathbf{f}_n = (f_{nx}, f_{ny})^T$, that is the local velocity projected on $\mathbf{u}_n$ is equal to:

$$\mathbf{f}_n = (v_n^T \cdot \mathbf{u}_n) \mathbf{u}_n. \quad (8)$$

We define the following quantities that will be used to estimate $\mathbf{v}_c$ and ω_c . They are updated in a similar way as the activity $\mathcal{A}$, as explained in section 3.2 :

$$\begin{cases} \Sigma_{\bar{f}_x}(t_n) = \Sigma_{\bar{f}_x}(t_{n-1})\lambda_n + f_{nx} \\ \Sigma_{\bar{f}_y}(t_n) = \Sigma_{\bar{f}_y}(t_{n-1})\lambda_n + f_{ny} \\ \Sigma_{\bar{f}_\omega}(t_n) = \Sigma_{\bar{f}_\omega}(t_{n-1})\lambda_n + \mathbf{f}_n^T \cdot \mathbf{o}_n \end{cases} \quad (9)$$

and we regroup these terms as well as the contour velocities in two vectors :

$$\Sigma \equiv \begin{pmatrix} \Sigma_{\bar{f}_x} \\ \Sigma_{\bar{f}_y} \\ \Sigma_{\bar{f}_\omega} \end{pmatrix}, \mathbf{V}_c \equiv \begin{pmatrix} v_{cx} \\ v_{cy} \\ \omega_c \end{pmatrix} \quad (10)$$

The following lines provide calculations details, leading to compact form of equation 16.

$$\begin{aligned} \mathbf{f}_n &= ((\mathbf{v}_c + \omega_c \mathbf{o}_n) \cdot (c_n, s_n)^T) (c_n, s_n)^T \\ &= \begin{pmatrix} c_n^2(v_{cx} - y_n\omega_c) + c_n s_n(v_{cy} + x_n\omega_c) \\ c_n s_n(v_{cx} - y_n\omega_c) + s_n^2(v_{cy} + x_n\omega_c) \end{pmatrix} \\ \mathbf{f}_n \cdot \mathbf{o}_n &= (v_{cx} - y_n\omega_c)(x_n c_n s_n - y_n c_n^2) \\ &\quad + (v_{cy} + x_n\omega_c)(x_n s_n^2 - y_n c_n s_n) \end{aligned} \quad (11)$$

In matrix form, it gives us

$$\begin{aligned} \mathbf{f}_n &= \begin{pmatrix} c_n^2 & c_n s_n & x_n c_n s_n - y_n c_n^2 \\ c_n s_n & s_n^2 & x_n s_n^2 - y_n c_n s_n \end{pmatrix} \mathbf{V}_c \\ \mathbf{f}_n \cdot \mathbf{o}_n &= \begin{pmatrix} x_n c_n s_n - y_n c_n^2 \\ x_n s_n^2 - y_n c_n s_n \\ x_n^2 s_n^2 + y_n^2 c_n^2 - 2x_n y_n c_n s_n \end{pmatrix}^T \cdot \mathbf{V}_c \end{aligned} \quad (12)$$

We define the matrix M such that:

$$M \equiv \begin{pmatrix} r_{cc} & r_{cs} & r_{xcs} - r_{ycc} \\ r_{cs} & r_{ss} & r_{xss} - r_{ycc} \\ r_{xcs} - r_{ycc} & r_{xss} - r_{ycc} & r_{xxss} + r_{yycc} - 2r_{xycc} \end{pmatrix}, \quad (13)$$

with $r_{\alpha\beta uv}(t_n) = r_{\alpha\beta uv}(t_{n-1})\lambda_n + \alpha_n \beta_n u_n v_n$, for $\alpha, \beta \in \{x, y\}$ and $u, v \in \{c, s\}$.

This matrix holds purely spatial information about the contour, this information being composed of the events locations $\mathbf{x}_n$ along with the local contour directions $\mathbf{u}_n$. From this definition of M , equation 12 can be combined into :

$$\begin{pmatrix} \mathbf{f}_n \\ \mathbf{f}_n \cdot \mathbf{o}_n \end{pmatrix} = (M(t_n) - M(t_{n-1})\lambda_n) \mathbf{V}_c. \quad (14)$$

And with definition given in equation 10, we obtain:

$$\begin{aligned} \Sigma(t_n) - \Sigma(t_{n-1})\lambda_n &= (M(t_n) - M(t_{n-1})\lambda_n) \mathbf{V}_c \\ \Leftrightarrow \Sigma(t_n) &= M(t_n) \mathbf{V}_c + \lambda_n (\Sigma(t_{n-1}) - M(t_{n-1}) \mathbf{V}_c). \end{aligned} \quad (15)$$

As $\Sigma(t=0) = \mathbf{0}$ and $M(t=0) = \mathbf{0}$, by recurrence, for all t_n , we have the main relation:

$$\Sigma = M \mathbf{V}_c \quad (16)$$

Hence, $\mathbf{V}_c$ is calculated by inverting M :

$$\mathbf{V}_c = M^{-1} \Sigma. \quad (17)$$

Both M and Σ are updated in a purely-event-based fashion. $\mathbf{V}_c$ contains the contour velocities in $\mathcal{R}_T$ that we want to nullify, as that means a correct tracking of the contour (illustration by Figure 4).

Figure 4: Description of the contour velocities $\mathbf{v}_c$ and ω_c estimation. We update the vector Σ and the matrix M defined in equations 10 and 13 using event location $\mathbf{x}_n$, its orthogonal vector $\mathbf{o}_n$, optical flow $\mathbf{f}_n$ and unit vector $\mathbf{u}_n = \mathbf{f}_n / \|\mathbf{f}_n\|$. All those vectors are expressed in $\mathcal{R}_T$.

3.5 Normalization for the tracker speed update

The estimated contour velocities $\mathbf{V}_c$ can be expressed as the opposites of errors vectors ϵ_v and ϵ_ω between the tracker velocities

$\mathbf{v}$ and ω , and the contour velocities $\mathbf{v}_{c,\mathcal{R}_f}$ and $\omega_{c,\mathcal{R}_f}$ in the local frame $\mathcal{R}_f$:

$$\begin{cases} -\mathbf{v}_c = \boldsymbol{\epsilon}_v \equiv \mathbf{v} - \mathbf{v}_{c,\mathcal{R}_f} \\ -\omega_c = \epsilon_\omega \equiv \omega - \omega_{c,\mathcal{R}_f} \end{cases} \quad (18)$$

We aim for an exponential dampening of that velocity error, leading to a match between the tracker and the contour velocities:

$$\begin{cases} \frac{\partial \mathbf{v}}{\partial t} = -\frac{\boldsymbol{\epsilon}_v}{\tau} = \frac{\mathbf{v}_c}{\tau} \\ \frac{\partial \omega}{\partial t} = -\frac{\epsilon_\omega}{\tau} = \frac{\omega_c}{\tau} \end{cases} \quad (19)$$

Let us define $\bar{\tau}$ the average time interval between two successive events, if $\bar{n}$ is the average number of events per second affecting that tracker, then :

$$\bar{n} = 1/\bar{\tau}. \quad (20)$$

Finally, we define $\delta \mathbf{v}_n$ and $\delta \omega_n$ the translation and rotation corrections applied to the tracker for the n -th event:

$$\begin{cases} \delta \mathbf{v}_n = \mathbf{v}(t_n) - \mathbf{v}(t_{n-1}) \\ \delta \omega_n = \omega(t_n) - \omega(t_{n-1}) \end{cases} \quad (21)$$

On average, $t_{n-1} = t_n - \bar{\tau}$, thus by differentiating these equations and using equations 19, we obtain

$$\begin{cases} \delta \mathbf{v}_n = \frac{\partial \mathbf{v}}{\partial t}(t_n) \bar{\tau} = \frac{\mathbf{v}_c \bar{\tau}}{\tau} \\ \delta \omega_n = \frac{\partial \omega}{\partial t}(t_n) \bar{\tau} = \frac{\omega_c \bar{\tau}}{\tau} \end{cases} \quad (22)$$

Let us consider now the evolution of the activity $\mathcal{A}$ defined in equation 4 between t and $t + \delta t$, with the constraint

$$\bar{\tau} \ll \delta t \ll \tau. \quad (23)$$

This constraint can be achieved, as $\bar{\tau}$ is the average time between two events, and τ is the time during which the tracker moves of one pixel : during τ , many events will occur, typically one for each pixel of the observed shape.

The exponential decay of the activity ensures that it reaches a steady state after some time. We approximate that the evolution of the activity over a restricted duration such as δt can be written as

$$\mathcal{A}(t + \delta t) \simeq \mathcal{A}(t) \rho(\delta t) + \Delta(\delta t) \quad (24)$$

with ρ being the decay term given by

$$\begin{aligned} \rho(\delta t) &= e^{-\frac{\delta t}{\tau}} \text{ with } \delta t \ll \tau \text{ giving} \\ \rho(\delta t) &\simeq (1 - \frac{\delta t}{\tau}) \end{aligned} \quad (25)$$

and Δ the sum of unit increases for each event that occurred between t and $t + \delta t$, given by

$$\begin{aligned} \Delta(\delta t) &= \sum_{k, t \leq t_k \leq t + \delta t} 1, \text{ with } \bar{\tau} \ll \delta t \text{ giving} \\ &\simeq \delta t \bar{n} \\ \Delta(\delta t) &\simeq \frac{\delta t}{\bar{\tau}} \end{aligned} \quad (26)$$

As activity reaches a steady state, we have

$$\begin{aligned} \mathcal{A}(t) &= \mathcal{A}(t + \delta t) \\ \mathcal{A}(t) &\simeq \mathcal{A}(t) (1 - \frac{\delta t}{\tau}) + \frac{\delta t}{\bar{\tau}} \\ \mathcal{A}(t) \frac{\delta t}{\tau} &\simeq \frac{\delta t}{\bar{\tau}} \\ \mathcal{A}(t) &\simeq \frac{\tau}{\bar{\tau}} \end{aligned} \quad (27)$$

Finally, injecting this result in 22, we obtain

$$\begin{cases} \delta \mathbf{v}_n = \frac{\mathbf{v}_c}{\mathcal{A}(t_n)} \\ \delta \omega_n = \frac{\omega_c}{\mathcal{A}(t_n)} \end{cases} \quad (28)$$

This is the speed correction used in our implementation, allowing for smooth speed update during the convergence process. The activity $\mathcal{A}$ acts as a normalization value for all quantities updated event-by-event in this work.

Figure 5 illustrates how the velocities correction mechanism operates "under the hood" of a tracker.

Figure 5: Convergence process of a tracker onto a simple feature. We separated translation - (a), (b) - and rotation - (c), (d) - for clarity purposes. In practise, both are resolved in parallel. (a) and (c) : the trackers starts moving as the features appears. Each event is associated with a local velocity $\mathbf{v}_n$ in red (cf. Figure 4) and allows to estimate the contour velocities in translation $\mathbf{v}_c$ and in rotation ω_c in grey. The corrections applied $\delta \mathbf{v}_n$ and $\delta \omega_n$ follow equation 28. In both translation and rotation cases, when the tracker speed matches the contour velocities - (b) and (d) -, the contour appears immobile in the tracker reference frame.

3.5.1 Aperture metric

The local and incremental approach in this work is prone to the same aperture problem as the optical flow it relies on. To solve this problem, we introduce a metric that measures the quality of a tracked structure i.e. if it allows to estimate the correct

velocity: a structure such as a single or a bundle of parallel lines is typically a bad candidate while intersecting lines or edges with finite curvature are good ones.

For clarity purposes, let us assume two local edges are captured by the tracker (the same analysis can be extended to n edges) and they are approximated as line segments with respective direction angles θ_1 and θ_2 . We define $\Delta\theta$ as the difference of these angles that we constrain to take value in $[0, \frac{\pi}{2}]$ (this does not remove any generality to the problem since we can always choose the smallest intersecting angle of the two lines for $\Delta\theta$). The edges define a good structure to track if $|\Delta\theta|$ is sufficiently large (i.e. lines less parallel). According to this hypothesis, the metric we need must be a monotonic function of $\Delta\theta$.

Let us express the unit direction vectors of the two line segments in their complex form $e^{i\theta_1}$ and $e^{i\theta_2}$. If we want to constrain the metric function to take value in $[0, 1]$, we can increase the vectors arguments by a factor of 2. Let $\mathbf{r}_1$ and $\mathbf{r}_2$ be these new unit vectors:

$$\mathbf{r}_1 = e^{i2\theta_1} \text{ and } \mathbf{r}_2 = e^{i2\theta_2}. \quad (29)$$

Assuming $\theta_2 \geq \theta_1$, we define the two quantities: $\bar{\theta} = \theta_1 + \theta_2$ and $\Delta\theta = \theta_2 - \theta_1$. Then the averaged resultant of the two vectors is:

$$\begin{aligned} \boldsymbol{\mu} &= \frac{e^{i(\bar{\theta}-\Delta\theta)} + e^{i(\bar{\theta}+\Delta\theta)}}{2} = \frac{e^{i\bar{\theta}}(e^{-i\Delta\theta} + e^{i\Delta\theta})}{2} \\ &= \cos(\Delta\theta)e^{i\bar{\theta}}. \end{aligned} \quad (30)$$

From here, we can see that we can build the metric function based on the cosine in the amplitude of $\boldsymbol{\mu}$. It is monotonic for $\Delta\theta \in [0, \pi/2]$, decreasing from 1 to 0. Thus, if the tracker captures two parallel edges, $\Delta\theta = 0$ and the function is maximum and if the two edges are orthogonal, $\Delta\theta = \frac{\pi}{2}$ and the function reaches its minimum of 0.

To build the metric function which will also be updated for each incoming event assigned to a tracker, we proceed as follows:

- from the set $\mathcal{E}_n$ defined in section 3.3, build the offset vector $\delta\mathbf{x}_k = \mathbf{x}_k - \bar{\mathbf{x}}_k$ and increase its direction angle by 2 to produce $\tilde{\delta\mathbf{x}}_k$.

These vectors are averaged over $\mathcal{E}_n$ into a vector giving the principal orientation of the local edge:

$$\bar{\mathbf{r}}_n = \frac{1}{N} \sum_{k \in \mathcal{E}_n} \tilde{\delta\mathbf{x}}_k, \quad (31)$$

- then we compute the temporal decaying sum of the normalized vectors $\bar{\mathbf{r}}_n$ and divide it by the tracker activity $\mathcal{A}(t_n)$:

$$\begin{cases} \mathbf{M}(t) = \mathbf{M}(t_{n-1})\lambda_n + \frac{\bar{\mathbf{r}}_n}{\|\bar{\mathbf{r}}_n\|} \\ \boldsymbol{\mu}(t_n) = \frac{\mathbf{M}(t_n)}{\mathcal{A}(t_n)} \end{cases} \quad (32)$$

The chosen metric is then $\|\boldsymbol{\mu}(t)\|$ which takes value in $[0, 1]$.

Structures subject to aperture ambiguities is filtered out by setting a threshold on $\|\boldsymbol{\mu}\|$ as good candidates to track yield small metric values.

3.5.2 Tracker Stabilization

Trackers are by default initialized with a null velocity. As events are registered, a tracker's motion parameters (velocity and position) are updated continuously, leading to a rapid convergence time. However, this continuous update might still report non null velocity errors and leaves us uncertain about the steady state of the trackers. We implement an additional estimator to lift this uncertainty based on the averaged norm of $\mathbf{x}_n - \bar{\mathbf{x}}_k$, as used in computing the optical flow in 6. As the tracking converges, the flow norm should converge to 0. We then build the ν function as follows:

$$\begin{cases} N(t_n) = N(t_{n-1})\lambda_n + \|\mathbf{x}_n - \bar{\mathbf{x}}_k\| \\ \nu(t_n) = \frac{N(t_n)}{\mathcal{A}(t_n)} \end{cases} \quad (33)$$

A threshold is set on ν (experimentally fixed to 0.5 pixel) below which we can state that a tracker has been stabilized. This value can be changed, but is not critical, as experiments have shown that as long as we stay within this order of magnitude, the described estimator is able to discriminate between a stabilization running and a converged state.

3.6 Time constant definition

So far, the mechanism of a tracker is built around the decay mechanism allowing it to be updated upon reception of an event and giving the tracker the ability to self-adjust to the scene dynamic. This decaying property is summarized by the time constant τ in the exponential function, hence the estimation of τ is one (if not the most), of the most important tasks to achieve. To estimate τ from the events, we are first considering the average squared velocity of points over the tracker area:

$$\|\overline{\mathbf{v}_n}\|^2 = \|\mathbf{v}\|^2 + 2\omega\mathbf{v} \cdot \overline{\mathbf{o}_n} + \omega^2 \|\overline{\mathbf{o}_n}\|^2, \quad (34)$$

where the averaged terms $\langle \mathbf{o}_n \rangle$ and $\langle \|\mathbf{o}_n\|^2 \rangle$ are updated according to the mechanism:

$$\begin{cases} \mathbf{O}(t_n) = \mathbf{O}(t_{n-1})\lambda_n + \mathbf{o}_n \\ R_{sq}(t_n) = R_{sq}(t_{n-1})\lambda_n + \|\mathbf{o}_n\|^2 \\ \overline{\mathbf{o}_n} = \frac{\mathbf{O}(t_n)}{\mathcal{A}_n} \\ \|\overline{\mathbf{o}_n}\|^2 = \frac{R_{sq}(t_n)}{\mathcal{A}_n} \end{cases} \quad (35)$$

Finally, we can define τ as

$$\tau(t_n) = \frac{1}{\sqrt{\|\overline{\mathbf{v}_n}\|^2}} \quad (36)$$

3.7 Spatial descriptor and tracker lock

For the entire tracking process, a rolling set of events is maintained from the events affecting the tracker. At time t , we only keep events that appeared between $t - N\tau(t)$ and t . Events from

Figure 6: Evolution of the main variables during the tracker convergence on a corner in translation, described in detail in section 3.8. Top graphs shows the velocity variable $\mathbf{v}$ of the tracker, the estimated contour velocity $\mathbf{v}_c$ w.r.t the tracker, and the updates for each event $\delta \mathbf{v}_n$. For clarity purposes, we use the norm for each of these variables. The second graph shows the evolution of the activity $\mathcal{A}$, the aperture metric $\mu = \|\mu\|$ and the convergence metric ν . Square pictures show the spatio-temporal context of the scene, while green circles contain the projected events, i.e the events from the tracker perspective. Key moments are highlighted with timestamps t_1 to t_5 .

set $\mathcal{E}_n$ are extracted from this spatial descriptor, stored as a $2 \times m$ matrix.

$$\mathbf{d} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_k^T \\ \vdots \\ \mathbf{x}_m^T \end{bmatrix} \quad (37)$$

This local descriptor is calculated in a continuous space i.e. the coordinates are real values instead of integers. This allows for a much smoother spatial representation of the object, often enhancing its visual appearance from the raw events, and can thus be used in following algorithms based on that tracking, such as feature matching, tracking recovery or object recognition.

3.7.1 Locking the feature

As the spatial descriptor so far is a rolling set, even with a tracker that has converged onto a trackable feature, a drift can still occur. To avoid that, we can *lock* the tracker.

With the tracker T going into the *locked* status at time t_l , we lock that spatial descriptor, and the new events are compared to the events stored in $\mathbf{d}$ at time t_l . This shape is then the reference shape, and the tracking process can remain stable for much longer times.

3.7.2 Disengaging the feature

One of the challenges of the approach is caused by sudden deceleration of tracked objects. As less events are produced during a deceleration the correction processes must be robust to rapid changes in dynamics. The solution lays in monitoring the activity $\mathcal{A}$ of the *locked* tracker. The idea being that below a certain threshold, one must *disengage* the tracker. We roll back its position $\mathbf{X}$ and angle θ to the last known values where a viable optical flow has been computed, and stop updating the position relative to the speed as described in 1. At the same time, we decay the speed itself, as only a rapid deceleration can induce such a behaviour :

$$\mathbf{v}(t_n) = \mathbf{v}(t_{n-1})\lambda_n \quad (38)$$

In most cases, this process provides sufficient time for the object to move again on the focal plane, and the tracking to resume, once the activity $\mathcal{A}$ is above a predefined threshold.

3.8 Illustration case

The main parameters of a tracker are highlighted in a practical case displayed in Figure 6. This real-data example shows a square moving in a pure translation movement at a constant speed of about $1000px.s^{-1}$. We show the temporal evolution of the six main parameters of a tracker. First, the norms of the tracker velocity $\|\mathbf{v}\|$, the norm of the contour velocity w.r.t the tracker $\|\mathbf{v}_c\|$ and the norm of the velocity variations for each event $\|\delta\mathbf{v}_n\|$. In a second graph are shown the activity $\mathcal{A}$, the aperture metric $\mu = \|\boldsymbol{\mu}\|$ and the convergence metric ν .

The sequence is broken into five key moments, each reflecting a different important phase of the process. At $t = t_1$, no event have occurred in the tracker region of interest (ROI) since its initialization. All tracker's parameters are either null or undefined, and the tracker is still. For $t = t_2$, the convergence process starts after the activity reaches a predefined threshold. The tracker then starts to cancel out any contour velocity. This results in a increase of the convergence metric μ . The aperture metric already shows that the structure captured by the tracker is a straight line, as it reaches a value close to 1. For clarity purposes, we deactivated the mechanism to reject tracking based on the aperture metric.

At $t = t_3$, the tracker reaches a steady state. All values remain almost constant, the tracker knows it has converged (ν is low) but the aperture issue remains. Only the orthogonal component of the velocity is known to be reliable, and the tracker is free to move along that line¹. During this phase, the contour velocity $\mathbf{v}_c$ - and thus the correction applied $\delta\mathbf{v}_n$ - are almost null.

Around $t = t_4$, a corner appears within the tracker ROI. Immediately, the aperture metric drops. The other corner edge feeds new velocity information to the tracker, resulting in an increase of the contour velocity $\mathbf{v}_c$ as well as the convergence metric ν . The tracker compensates this velocity to cancel out this remaining velocity. One can notice a small increase of the activity. This is due to an increase of the number of events assigned to the tracker when the corner appears, while the velocity - and thus the time constant - does not immediately change.

Finally, at $t = t_5$, once the correction has been applied, the tracker speed matches the corner speed, and the features appears immobile from the tracker perspective (in the tracker reference system $\mathcal{R}_T$). Both the aperture and the convergence metrics are below their predefined threshold, and the shape is *locked* as described in section 3.7.1. This increases even more the stability of the tracking. As the tracker velocity increases, its time constant τ lowers, slightly lowering the activity, and improving the time response of the whole system to match the scene dynamics.

The tracker has been intentionally initialized far from its target feature resulting in a $100ms$ convergence time. When trackers

were initialized closer to the target for the very same example, the convergence time was on average less than $10ms$.

3.9 Algorithms

The method presented so far is implemented as summarized by the algorithm detailed in this section. We can decompose the tracking into three blocks. All tracker variables previously described are now indexed by i :

Algorithm 1 Main Algorithm

---

require: N_T trackers initialized in the focal plane.

```

for each  $e_n = \{\mathbf{X}_n, t_n\}$  do
  for Tracker  $T_i, i \in [1, N_T]$  do
    Update  $T_i$  according to algorithm 2
    if  $\|\mathbf{X}_n - \mathbf{X}_i\| < R$  then
      Compute  $\mathbf{x}_n$  according to eq. 2
      Compute velocity error according algorithm 3

```

Algorithm 2 Update for each event

---

require: Tracker T_i , event $e_n = \{\mathbf{X}_n, t_n\}$.

Compute time elapsed since last event Δt_n

Update tracker position :

$$\mathbf{X}_i(t_n) \leftarrow \mathbf{X}_i(t_{n-1}) + (t_n - t_{n-1})\mathbf{v}_i(t_{n-1})$$

$$\theta_i(t_n) \leftarrow \theta_i(t_{n-1}) + (t_n - t_{n-1})\omega_i(t_{n-1}).$$

if $\mathbf{X}_i$ is out of screen **then**

Reset tracker T_i

$T_i \leftarrow Idle$

return

Compute Exponential decay:

$$\lambda_i(t_n) \leftarrow \exp\left(-\frac{t_n - t_{n-1}}{\tau_i(t_{n-1})}\right)$$

Update tracker monitoring variables with $\lambda_i(t_n)$:

$$\mathcal{A}_i(t_n) \leftarrow \lambda_i(t_n)\mathcal{A}_i(t_{n-1}).$$

if $\mathcal{A}_i(t_n) < \mathcal{A}_{reset}$ **then**

Reset tracker T_i

$T_i \leftarrow Idle$

return

else if $\mathcal{A}_i(t_n) < \mathcal{A}_{disengage}$ **then**

$T_i \leftarrow Disengaged$

Reset position and angle to last valid values

else if $T_i = Disengaged$ & $\mathcal{A}_i(t_n) \geq \mathcal{A}_{disengage}$ **then**

$T_i \leftarrow Locked$

We define for each tracker a set of important states that are used in the implementation². Those states are:

- *Idle*: a tracker is considered *Idle* as long as the amount of events assigned to the tracker is under the experimentally predefined threshold.
- *Stabilizing*: it is considered *Stabilized* when computation is being performed but the convergence estimator is still too high. This means that the tracker still needs time to catch up with the target feature.
- *Converged*: a tracker has *Converged* once it has stabilized and matched the target feature.

1. In most cases, the tracker has a null velocity component in the direction of the line. In our case, this induces a negative velocity along the y axis, and the tracker will move towards the bottom of the focal plane

2. The algorithm implementation is accessible on <https://github.com/neuromorphic-paris/eb-tracker/>.

Algorithm 3 Update for events affecting the tracker**require:** Tracker T_i , event $e_n = \{X_n, t_n\}$. $\mathcal{A}_i(t_n) \leftarrow \mathcal{A}_i(t_{n-1}) + 1$ **if** $T_i = \text{Idle}$ & $\mathcal{A}_i(t_n) > \mathcal{A}_{\text{start}}$ **then** $T_i \leftarrow \text{Stabilizing}$

Compute local flow, cf equation 6

Update dynamical estimator, cf equations 9 to 13

Compute contour velocity, cf equation 17

Compute speed correction, cf equation 28

Update aperture estimator, cf equation 32

Update convergence estimator, cf equation 33

Update tracker state, cf Figure 7

Update time constant, cf equation 36

Figure 7: Relations between the different features' states. We show here the key conditions for the evolution of a tracker, from its initialization in an *Idle* state to its reset. Conditions leaving the tracker state unchanged are not shown here for clarity purposes.

- *Locked*: this state maintains the tracker on tracking a structure for which the algorithm detects no aperture problem i.e. it is valid only when *Aperture_issue* is "false".

- *Aperture_issue*: a tracker has *Aperture_issue*="true" if it detects an aperture problem from the aperture metric. This status prevents the tracker from being *Locked* to the target even if it is in a *Converged* state.
- *Disengaged*: A tracker has *disengaged*="true" when it is in a *locked* state but its activity is too low. The position is no longer updated using the speed, and the speed is decayed, until new events appear again.

Those states are shown in Figure 7, with their corresponding relations and the parameters required to update those states.

Table 1 shows the experimentally predefined values for each state threshold that were used in all experiments. Note that some of these values may depend linearly on the radius R of the tracker. This radius must especially be changed accordingly to the sensor resolution.

Name	Value	Proportional to R
R	10	Yes
$\mathcal{A}_{\text{start}}$	40	Yes
$\mathcal{A}_{\text{disengage}}$	10	Yes
$\mathcal{A}_{\text{reset}}$	4	Yes
κ_{aperture}	0.55	No
κ_{conv}	0.6	No

Table 1: Experimental values for the hyperparameters presented. Most of them are either constant across all experiment, or scale linearly with the radius R .

At this stage, the proposed algorithm is able to track structures that are not restricted to corners or simple edges and can be somehow complex as shows the sample given by Figure 8. These features were extracted during a run of our algorithm on a recording of a city environment. This unconstrained structure selection allows to have a wide range of features that are crucial to higher level tasks.

4 EXPERIMENTS AND PERFORMANCES EVALUATION

We compare the algorithm performances to the state-of-the-art. As shown in section 3, the tracking and detection mechanisms are interlaced such that features emerge in an online manner. This is a purposely designed property because users do not have to assume complex priors on the features to track. We tested the algorithm on sequences generated by the event-based camera simulator, alongside with the provided dataset, acquired by event-based sensors in natural environments [19]. To assess the algorithm performances, the provided frames are used and a classical frame-based tracking algorithm is initialized when a tracker is *locked*. The same dataset provides ground-truth from an Inertial Measurement Unit (IMU), allowing us to assess directly the camera's ego-motion inferred from the trackers estimated velocities. The data is acquired using a DAVIS 240C, with a resolution of 240×180 pixels [9]. Preliminary tests have also been performed with a 640×480 VGA ATIS [18]. Similar results have been found using both sensors relying on the same experimental set of parameters. The algorithm is bench-marked using a python implementation. We initialized 63 trackers dispatched on a 9×7 grid, with 25 pixels diameter ROI.

4.1 Comparison with Frame-based tracking

When frames are available, we can compare the event-based tracker with a frame-based algorithm. For this purpose, we have used the frame-based Channel and Spatial Reliability Tracking (CSRT), an OpenCV implementation of [25] as the baseline. Each event-based tracker, on locking, is compared to a CSRT tracker initialized at the corresponding location on the temporally closest frame. We compare

the two trackings in stages where the event-based tracker is in *locked* status.

The CSRT results are visually checked for the entire duration of the benchmark. Even if the CSRT is considered as one of the state-of-the-art frame-based tracking algorithm, it does occasionally fail. In order to have the most accurate benchmark assessment for the proposed algorithm, we manually removed the failing CSRT trackers scenarios although the presented event-based tracker has been performing correctly.

For certain highly textured scenes the frame-based tracker was also unable to provide any ground-truth data due to excessive displacement between two consecutive frames and also because of motion blur. However, this method allowed to automatize the benchmark and to avoid a manual labeling of considered scenes.

4.2 Benchmark with the Event Camera Simulator

We used data generated with the Event Camera Simulator[19] for which the ground-truth can be inferred from a known camera motion and scene depth. Simple movements mixing translation and axial rotation have been simulated, along with the corresponding frames. For each tracker *locked*, we determine the 3D location of the object and re-project its location on the focal plane. We generated two sequences. The first one uses the shapes poster from the Event-Camera dataset, while the second one uses a texture of a book cover. The event-based and frame-based algorithms use similar window sizes: a 25px ROI for both the event-based method and the CSRT trackers. The trackers accuracy and life-span are reported in Table 2.

Sequence	Valid Tracks		Asserted Error (px.)		Asserted Lifespan (s)	
	this method	CSRT	this method	CSRT	this method	CSRT
shapes (short)	93.0%	76.3%	1.40	2.29	1.06	1.06
shapes (long)	89.9%	92.8%	1.36	1.90	1.57	1.71
book (short)	92.6%	60.1%	1.28	2.02	0.68	0.60
book (long)	91.5%	81.4%	1.05	2.52	1.02	1.09

Table 2: Comparison between the frame-based tracking and the event-based tracker using simulated data. CSRT trackers are removed once the event-based tracker has lost the target due to the disappearance of the target from the scene.

4.3 Benchmark on event camera dataset

The performance comparison is now applied the same sequences of the dataset that were reported in previous work in [8]. Since no ground-truth is available, the CSRT trackers were used as a ground-truth for comparison. We report in Table 3 the results obtained for six sequences. A track is considered *valid* if its mean error to the ground-truth is less than 5 pixels. We also display the resulting mean error of those valid tracks and their average lifespan. Most of the scene’s objects come in and out of the field of view of the camera, tracker that go out of the focal plane are considered lost. This explains the average lifespan of the tracker trackers. Re-entry strategy are beyond the scope of the paper as they imply storing previously seen trackers. Results show lifespans of several seconds with accurate tracking.

4.4 Panoramic reconstruction

Using a set of trackers we will now compute a panoramic reconstruction of an outdoors scene using a recorded sequence of 6.6s. Each *green* rectangle represents a tracker used for that reconstruction. We display on that panoramic view the most representative set of trackers that were *locked* and used for that reconstructions (*blue dots*) with some of them highlighted in colored circles for clarity purposes, and shown in more details below the

Stream	Duration	Valid Tracks	Error [px]	Lifespan
shapes_rotation	30s	71.9%	1.90	0.60s
shapes_translation	30s	93.7%	1.66	0.84s
shapes_6dof	30s	86.8%	1.90	0.61s
poster_rotation	11s	69.1%	1.85	0.69s
poster_translation	14s	68.1%	2.12	0.88s
poster_6dof	20s	62.1%	2.23	0.49s

Table 3: Results on GT comparison with CSRT trackers in OpenCV library.

panoramic view.

This selection shows the wide variety of features the trackers can lock on. Below those features are displayed the average reconstruction error ϵ and the duration Δ_t during which the tracker stayed *locked* on its feature. We notice that the longer duration of tracking reported concerned mainly highly “complex” features. The top right feature is displayed here to show the limitation of the algorithm under textured and noised recordings : the trackers can sometimes *lock* onto ill-defined shapes, resulting in poor tracking duration. However, in these cases, the convergence estimator previously described and the tracker activity monitoring are able to cancel this tracker, noticing that it does not “behave” as it should.

For this online robust reconstruction, we used 471 trackers, with an average tracking time of 0.62s, and an average reconstruction error of 3.3 pixels considering all trackers. For the non outliers the median standard deviation of the reconstruction error is 0.72 pixels. This means that 72.7% of the trackers that were selected could successfully track features with an error of less than 1 pixels. Of those 471 trackers, 27.3% outliers reached the reconstruction error threshold of 20 pixels. In most of the cases, this was due to “blurred” straight lines that can sometimes report no aperture issue when small rapid saccadic motions occur.

4.5 Comparison with groundtruth with IMU data

The Event-Camera Dataset [19] provides IMU and camera pose data, serving as ground-truth measurements. This information is particularly relevant for the 3 degree of freedom ego-motion of the camera. Assuming $M = [X, Y, Z]^T$, $m = [x, y, z]^T$ being a 3D point and its projection into the focal plane in the camera coordinate frame, the instantaneous velocities are given by:

$$\begin{cases} \dot{M} = \omega \times M + t, \\ \dot{m} = \frac{d(\frac{f}{Z}M)}{dt} = \frac{f}{Z}\dot{M} - \frac{\dot{Z}}{Z}m \end{cases} \quad (39)$$

where ω and t are respectively the angular and the linear velocities of the camera. Since the velocity in the z axis of m is null, this is equivalent to:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -\frac{xy}{f} & f + \frac{x^2}{f} & -y & \frac{f}{Z} & 0 & \frac{x}{Z} \\ -\frac{y^2}{f} - f & \frac{xy}{f} & x & 0 & \frac{f}{Z} & -\frac{y}{Z} \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \\ t_x \\ t_y \\ t_z \end{pmatrix} \quad (40)$$

This flow is related to the optical flow calculated in the focal plane via the upper left 2×2 submatrix k , taken from the intrinsic matrix of the camera:

$$k^{-1} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}. \quad (41)$$

Equation 40 can be solved for ω knowing only the calibration parameters and if the linear translation in the ego-motion is null. We are

Figure 8: A Panorama reconstruction of a natural outdoors scene using a hand-held event-based camera (shown in A using a conventional frame-camera for clarity purposes). In B, the constructed panorama from events emphasizing a subset of the features used for clarity. Some highlighted features are represented with colored circles to show the diversity of what can be tracked (shown in C) with their average reconstruction error ϵ and the duration Δ_t during which a tracker stayed *locked* on its feature. Green squares shown in B, show some of the focal plane locations used to construct the panorama.

focusing on that particular case because to solve for the complete 3D velocity, one needs to provide a set of M and their projection m and this is not available in the Event-based Camera Dataset. We show the estimated ego-motion results for the sequence `shapes_rotation` in Figure 9. The axis have been normalized, assessing for x and y rotations a geometric constant of $220 \text{ pixels.rad}^{-1}$. For the z -axis, the normalization constant found was 0.98, showing the good match between this ego-motion computation and the ground-truth provided.

4.6 Trackers stability and latency

By design, the tracker time constant τ is auto-adjusted during the tracking w.r.t. the scene dynamics and it is the only parameter that needs explicit initialization as the inverse of a minimal speed, experimentally chosen as about 5 pixels.s^{-1} . The inertial behavior of a tracker is implemented to ensure a smooth and a robust to noise tracking mechanism. A tracker loses its targets because it is getting out of the camera’s field of view, rather than because of drifts caused by a non robust tracking approach.

The compromise to the tracker stability is the delay caused by the inertia. This is a non desired effect for a low latency vision sensor or for any vision sensor’s in general. However, the velocity based tracking algorithm we introduce has an interesting property that we can observe experimentally: on average, a tracker converges to the correct location within an radius of 20 to 30 pixels, when a structure enters the receptive field of the tracker.

This typical constant offset for the tracker to converge is an important asset as the time to convergence is a decreasing function of the speed i.e. the faster the apparent motion, the faster the tracker converges, hence the lower the latency of the algorithm.

5 CONCLUSION

This work introduced an event-based method to select and track simultaneously local spatial structures from a stream of events output from an event-based camera in real-time. The presented method relies on a velocity based tracking approach to converges towards selected features from which the true velocity can be estimated. More important, it is designed to be resilient to the aperture problem. The algorithm is also designed in a way to reduce the number of parameters to be set by users: mainly a decay coefficient that represents a dimensionless inertia parameter. The method is expected to work even better for higher spatial resolutions, as higher spatial resolution combined with the current sub-millisecond temporal precision implies that velocity can be estimated more accurately.

The method is showing robustness in a wide range of conditions. It can detect and track a large variety of spatial structures at different velocities. Tracking benchmarks have shown that it is accurate while being updated event by event, at the native rate driven by the scene thus allowing to reduce resources requirements. This has been made possible by estimating reliable velocities for each tracked feature. Most importantly, using several projection velocities for initialization allowed to have very little dependency on the tuning parameters.

Figure 9: Comparison of tracking results for *shapes_rotation* camera's estimated angular velocity. Trackers velocities are averaged to extract the position and speed of an initially centered point, assuming the objects on the scene are static and only the camera moves. *Red lines*: Gyroscopic data from the IMU provided with the DAVIS dataset [19]. *Blue lines*: Estimated rotational speed for each axis of the gyroscope. *Green lines*: Error between the reconstructed rotation and the IMU ground-truth. *Top graph*: Vertical rotation of the camera, producing an horizontal speed of the objects. Mean error : 0.04rad/s . *Center graph*: Horizontal rotation of the camera, producing a vertical speed of the objects. Mean error : 0.005rad/s . *Bottom graph*: Rotation of the camera along the optical axis. Mean error : 0.01rad/s . *Top images*: Snapshots from the corresponding scenes, with a restricted sample of the tracked features and their respective velocities orientations.

From a larger perspective, this paper introduces a new canonical general scheme for computation using event-based cameras that allows for an independence from motion direction and velocity changes. The introduction of a simultaneous use of space and precise timing information brought by each incoming event with a combined regularization scheme using the local activities of events proved to be efficient for local computations. The same scheme can be applied to a wide variety of problems where the same approach could be applied efficiently.

REFERENCES

- [1] C. Harris and M. Stephens, "A combined corner and edge detector," *Proceedings of the 4th Alvey Vision Conference*, pp. 147–151, 1988. **1**
- [2] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision*. Graz, Austria, 2006, pp. 430–443. **1, 2**
- [3] X. Clady, S. Ieng, and R. Benosman, "Asynchronous event-based corner detection and matching," *Neural Computation*, vol. 66, pp. 91–106, 2015. **2**
- [4] V. Vasco, A. Glover, and C. Bartolozzi, "Fast event-based harris corner detection exploiting the advantages of event-driven cameras," in *International Conference on Intelligent Robots and Systems*. IEEE, 2016, pp. 4144–4149. **2**
- [5] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 2, pp. 407–417, Feb 2014. [Online]. Available: <http://dx.doi.org/10.1109/tnnls.2013.2273537> **2, 3**
- [6] E. Mueggler, C. Bartolozzi, and D. Scaramuzza, "Fast event-based corner detection," in *British Machine Vision Conference*, 2017. **2**
- [7] I. Alzugaray and M. Chli, "Asynchronous corner detection and tracking for event cameras in real time," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, p. 3177–3184, Oct 2018. [Online]. Available: <http://dx.doi.org/10.1109/ra.2018.2849882> **2**
- [8] —, "Ace: An efficient asynchronous corner tracker for event cameras," *2018 International Conference on 3D Vision (3DV)*, Sep 2018. [Online]. Available: <http://dx.doi.org/10.1109/3dv.2018.00080> **2, 10**
- [9] C. Brandli, R. Berner, M. Yang, S. C. Liu, and T. Delbruck, "A240x180 130db 3 μs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014. **2, 9**

- [10] D. Tedaldi, G. gallego, E. Mueggler, and D. Scaramuzza, "Feature detection and tracking with the dynamic and active-pixel vision sensor (davis)," in *International Conference on Event-based Control, Communication, and Signal Processing. Krakow, Poland.* IEEE, 2016. 2
- [11] D. Gehrig, H. Rebecq, G. Gallego, and D. Scaramuzza, "Asynchronous, photometric feature tracking using events and frames," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. 2
- [12] A. Z. Zhu, N. Atanasov, and K. Daniilidis, "Event-based feature tracking with probabilistic data association," in *International Conference on Robotics and Automation. Singapore.* IEEE, 2017. 2
- [13] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image Vision Comput.*, vol. 10, no. 3, pp. 145–155, Apr. 1992. [Online]. Available: [http://dx.doi.org/10.1016/0262-8856\(92\)90066-C](http://dx.doi.org/10.1016/0262-8856(92)90066-C) 2
- [14] G. Gallego, H. Rebecq, and D. Scaramuzza, "A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation," 2018. 2
- [15] C. Posch, D. Matolin, and 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*, vol. 46, no. 1, pp. 259–275, Jan 2011. [Online]. Available: <http://dx.doi.org/10.1109/jssc.2010.2085952> 2
- [16] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120 db 15 μ s latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008. 2
- [17] M. Guo, J. Huang, and S. Chen, "Live demonstration: A 768x640 pixels 200meps dynamic vision sensor," in *IEEE International Symposium on Circuits and Systems. Baltimore, MD, USA.* IEEE, 2017. 2
- [18] C. Posch, D. Matolin, and R. Wohlgenannt, "A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds," *J. Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2011. [Online]. Available: <http://dblp.uni-trier.de/db/journals/jssc/jssc46.html#PoschMW11> 2, 9
- [19] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam," *The International Journal of Robotics Research*, vol. 36, no. 2, p. 142–149, Feb 2017. [Online]. Available: <http://dx.doi.org/10.1177/0278364917691115> 2, 9, 10, 12
- [20] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240 Å 180 130 db 3 Ås latency global shutter spatiotemporal vision sensor," *Solid-State Circuits, IEEE Journal of*, vol. 49, pp. 2333–2341, 10 2014. 2
- [21] X. Clady, S.-H. Ieng, and R. Benosman, "Asynchronous event-based corner detection and matching," *Neural Networks*, vol. 66, pp. 91–106, 2015. 3
- [22] P. Liu, M. R. Lyu, I. King, and J. Xu, "Selfflow: Self-supervised learning of optical flow," *CoRR*, vol. abs/1904.09117, 2019. [Online]. Available: <http://arxiv.org/abs/1904.09117> 3
- [23] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow." 3
- [24] H. Akolkar, S. Ieng, and R. Benosman, "See before you see: Real-time high speed motion prediction using fast aperture-robust event-driven visual flow," *CoRR*, vol. abs/1811.11135, 2018. [Online]. Available: <http://arxiv.org/abs/1811.11135> 4
- [25] A. Lukezic, T. Vojir, L. Cehovin Zajc, J. Matas, and M. Kristan, "Discriminative correlation filter with channel and spatial reliability," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 9