ACE: An Efficient Asynchronous Corner Tracker for Event Cameras

Ignacio Alzugaray and Margarita Chli
Vision for Robotics Lab, ETH Zürich

Abstract

The emergence of bio-inspired event cameras has opened up new exciting possibilities in high-frequency tracking, overcoming some of the limitations of traditional frame-based vision (e.g. motion blur during high-speed motions or saturation in scenes with high dynamic range). As a result, research has been focusing on the processing of their unusual output: an asynchronous stream of events. With the majority of existing techniques discretizing the event-stream into frame-like representations, we are yet to harness the true power of these cameras.

In this paper, we propose the ACE tracker: a purely asynchronous framework to track corner-event features. Evaluation on benchmarking datasets reveals significant improvements in accuracy and computational efficiency in comparison to state-of-the-art event-based trackers. ACE achieves robust performance even in challenging scenarios, where traditional frame-based vision algorithms fail.

Video – https://youtu.be/I31yQqmCsfs

1. Introduction

The increased interest in robotic perception over last decades has been largely fuelled by the success of Computer Vision algorithms employing visible-light frame-based cameras. Such conventional cameras are able to capture images with high information content about the camera's surroundings and have become most popular in a variety of applications due to their low cost, compactness, and ubiquity. The ability of estimating jointly the egomotion of the camera and its surroundings, commonly referred to as Simultaneous Localization And Mapping (SLAM), represents one of the most important milestones in the field.

Conventional frame-based cameras, however, exhibit several limitations by design; they capture images of the scene at fixed rate, regardless of the amount of new information present in each new image. As a consequence, the incoming information is most often highly redundant, wasting precious computational resources to process it. On the other hand, highly dynamic scenes or camera motion pose great difficulties to frame-based trackers as motion blur and insufficient overlap across consecutive frames result to lack of sufficient information to maintain tracking.

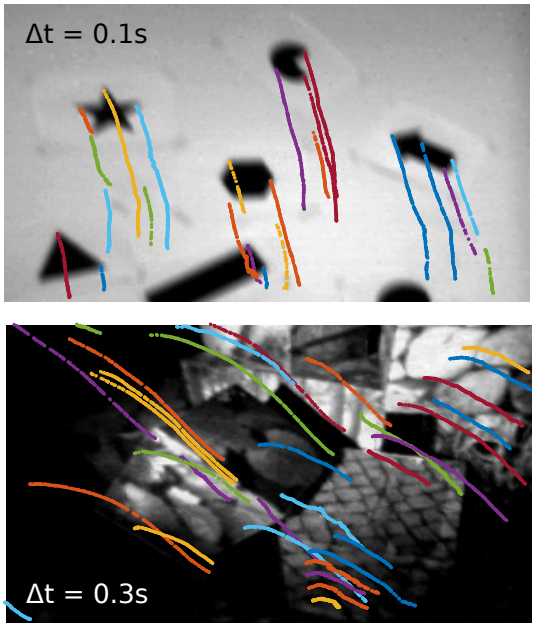

Figure 1. The proposed ACE tracker is able to retrieve the corner-event tracks in the scene even under high-dynamic motion (top) or in scenes with high dynamic range of illumination (bottom), tested on the dataset of [14]. Colors represent individual corner tracks shown over a time-window Δt. The ACE tracker operates directly on the event-stream, but the corresponding (blurry) intensity images are superimposed here to provide context.

Event cameras or Dynamic Vision Sensors (DVS) [10, 15, 3] promise to overcome some of these issues in traditional frame-based vision and, thus, have captured the attention of the community. Enabling each pixel of an event camera to react independently to illumination changes, a change in illumination at a particular pixel over a specific threshold triggers an event at that pixel with a temporal resolution of the order of μs. Such events are recorded asynchronously with their corresponding pixel locations and timestamps in the event-stream, effectively compressing all intensity changes during the camera's trajectory. The reduction of the visual experience to an event-stream sums up to a significantly lower data bandwidth than image-frames, exhibiting higher dynamic range and lower power consumption at the same time.

On the quest to fulfil the great promise of event cameras, the research community is driven to revisit the fundamentals of image processing in order to process the event-stream, which violates the traditional frame-based time discretization. Moreover, as events merely register incremental changes, the absolute intensity values of the scene are no longer directly observable. As a consequence, some of the most basic operations, such as data association for salient-region matching, are compromised.

Inspired by the recent advances in asynchronous feature detection and in particular, corner-event detectors [1, 4, 12, 20] the contributions of this paper are the following:

1. a novel and generic region descriptor for event data, applicable to any event-based algorithm, and

2. the Asynchronous Corner-Event (ACE) Tracker that employs these descriptors to establish local correspondence between corner-events and retrieves corner tracks asynchronously and efficiently.

2. Previous Work

The promising characteristics of event cameras have lead to the development of several SLAM pipelines employing such sensors over the last few years. Early approaches [21, 7, 16] avoid the explicit definition of features and thus the data association problem. Exploiting the natural sensitivity of event cameras to high contrast edges, these approaches rely on edge maps representations of the scene in which they track the camera pose. Most modern event-based SLAM pipelines, however, detect and track corner features. For instance, Rebecq *et al*. [17] introduce a method to compensate for the motion of the camera in the captured events within a predefined window using an Inertial Measurement Unit (IMU). Forming artificial frame-like edge images by accumulating events, they detect FAST corners [18] and track them with the KLT tracker [11]. Despite its impressive performance, this approach does not exploit the asynchronicity of events, but only their high temporal resolution, operating as a conventional frame-based SLAM at high-frequency.

Zhu *et al*. [23] propose a similar approach, in which the motion of the camera is compensated locally, rather than globally, and corner features are tracked by means of an expensive Expectation Maximization (EM) scheme that estimates the optical flow, without requiring IMU cues. In contrast to [17], this approach considers an adaptive window of events between tracking iterations instead of a fixed one. The size of the event-window is estimated online based on the optical flow, [13, 2], i.e. the rate of visual change of the scene. This corner tracking method is developed into a full SLAM pipeline in [22], in which it is combined with inertial cues from an IMU.

Establishing a window of events between tracking iterations, of either fixed [17] or adaptive [22] size, these methods impose time-discretizations as in frame-based SLAM, still ignoring the asynchronicity of the events. Nonetheless, there exist several works able to detect corner features asynchronously as new events are generated, classifying them as corner-events. Inspired by frame-based techniques, [20] proposes a corner-event detector inspired by the Harris detector [6], and [12] by FAST corners [18]. Alzugaray and Chli [1] propose a more efficient method to detect corner-events under generic camera motions and introduce a simple corner-event tracker based on proximity of detections in the image space. Clady *et al*. [4] propose a corner-event detector based on the estimated local optical flow [2], in addition to a simple data association algorithm, establishing correspondences across corner-events based on their velocity and predicted position under the assumption of smooth motion, tested on simple and trivial real scenes.

More complex data association schemes have been proposed, but mainly aimed to high-level tasks, such as pattern recognition [8, 9, 19, 5], where prior training to a model is required. Therefore, these methods are not suitable for on-line corner-event tracking, where no *a priori* knowledge of the scene can be assumed. In contrast to other existing works, this paper proposes a robust and efficient framework to establish correspondences between corner-events, effectively retrieving asynchronous corner tracks without requiring from prior knowledge of the scene or camera motion.

3. Event Cameras and Corner-Events

Let $I(x, y, t)$ be the log-intensity value measured at the pixel location (x, y), where an event triggered at time t. A new event e is generated if, after an arbitrary period of time Δt, the absolute difference of I reaches a specific threshold K. Formally,

$$\Delta I(x, y) = I(x, y, t + \Delta t) - I(x, y, t) = pK \ , \qquad (1)$$

where p denotes the event's polarity (i.e. is either 1 or -1) to indicate whether I increases or decreases. Ideally, a new event $e = \{t, x, y, p\}$ is generated as soon as this condition is met, and appended asynchronously to the event-stream. The rate of generation new events is thus directly correlated to the amount of visual change in the scene.

Employing conventional frame-based cameras, the location of corner features are detected at discrete times, matching the frame-rate. In event cameras, however, the location of a corner feature can be detected as soon as it changes to a new pixel location and generates a new event. As a result, the resulting event-based corner track can be more finely defined in time and space, regardless of how fast the feature moved in the image plane, as depicted in Fig. 2. The proposed ACE tracker is agnostic to the employed corner-event

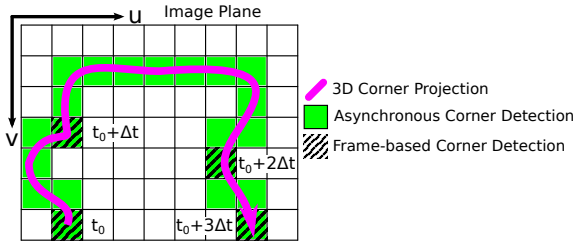

Figure 2. Corners in the 3D scene project to a continuous trajectory (magenta) in the image plane. In frame-based vision, the position of the corner in the image plane can be detected at discrete times (striped pixels), whereas event cameras enable their asynchronous detection, ideally, as soon as they trigger an event in a new pixel location (green pixels).

detector, only requiring asynchronous detections that can be finely tracked in the image space.

4. Local Region Descriptor from Events

To establish correspondences between detected corner-events, we define a new efficient descriptor that capture the local spatio-temporal context of each corner-event. Ideally, this descriptor would capture the information from all previously triggered events in the stream. Since processing all the previous events each time a new corner-event is detected would not be scalable, we employ a Surface of Active Events (SAE) [2] to summarize the event-stream at each instant. In brief, the SAE maps each pixel location in the image array to the timestamp of the latest event (not necessarily a corner-event) triggered in such location. Essentially, the SAE defines a dynamical, parameter-free temporal window in the event-stream that is updated asynchronously.

Upon the generation of a new event (or more specifically, a corner-event in this paper), we examine a local patch in the SAE surrounding the event's pixel location. The SAE patch contains only absolute temporal information and thus it is not directly comparable to other SAE patches captured at different time instants as a region descriptor. Instead, we map the absolute timestamps from the local SAE patch into a descriptor patch D of the same size, in which the values are normalized between $[0, 1]$.

4.1. Descriptor Normalization

The distribution of timestamps in the SAE is intrinsically related to the way event cameras perceive the visual scene. The locations in the SAE with newest timestamps are those where the high intensity contrast regions, i.e. mainly edges, of the scene are currently projected in the image plane, whereas the locations with older timestamps represent where they were previously projected (For simplicity, we refer to these high intensity contrast regions as edges for the rest of the paper). As we assume no prior knowledge about the camera and scene motion, we must rely on the locations of the SAE with newest timestamps to compute our descriptor, i.e. the ones where the edges from the scene are most probably projected at each instant. For this purpose, we advocate for an inexpensive normalization approach of the local SAE patch to derive the proposed descriptor, which retains the relative importance of the locations in the patch according to their timestamps. In the following paragraphs we analyse the suitability of different normalization techniques which are illustrated in Fig. 3.

A time-window normalization, i.e. mapping the SAE timestamps between $[0, 1]$ to those values within a temporal window and 0 otherwise, is unsuitable for event cameras unless the motion dynamics of the camera and/or scenes are *a priori* known or can be estimated. A too large time-window, according to the instantaneous motion dynamics, diminishes the relative importance of the current edges' position in the descriptor with respect to the previous edges' position. Conversely, a too small time-window may diminish the importance of locations where the edges are currently located.

A min-max normalization, i.e. mapping the SAE timestamps $[0, 1]$ between the oldest and newest values, render, in most of the cases, the current edges' position indistinguishable from positions of previously detected edges. Note that, in the local SAE, only a fraction of the patch is determined by the previous locations of the currently captured edges. The other (older) parts of the SAE depends on the past detected positions of other edges, which can be arbitrarily old.

In the computation of the proposed descriptors, we employ Sort Normalization, sorting the timestamps of the local SAE in an increasing order, and assigning the corresponding relative order to each location in the descriptor, normalized to be in the range $[0, 1]$. As a result, we can capture the relative importance of the different parts of the patch in our descriptor, without requiring an experiment-dependent definition of a time-window, while also exhibiting robustness to arbitrarily old locations in the local SAE.

4.2. Descriptor Distance

We measure the distance of descriptors by the amount of overlap between of their normalized local patch. The distance d between the descriptor patch D_A and D_B is formally as follows:

$$d(D_A, D_B) = 1 - \frac{\sum \min(D_A, D_B)}{\max\left(\sum D_A, \sum D_B\right)} \in [0, 1] \ , \quad (2)$$

where $\min(D_A, D_B)$ is an element-wise operation that results in the so-called overlapping descriptor $D_{A,B}$. Summing the elements in each pixel location of the overlapping descriptor, effectively the area under $D_{A,B}$, we compute the distance d. The divisor acts as a normalization term.

Using the sum of elements of $D_{A,B}$ in the computation of d, we favour (contributing to a smaller d) the newer locations of the descriptors that are common to both D_A and

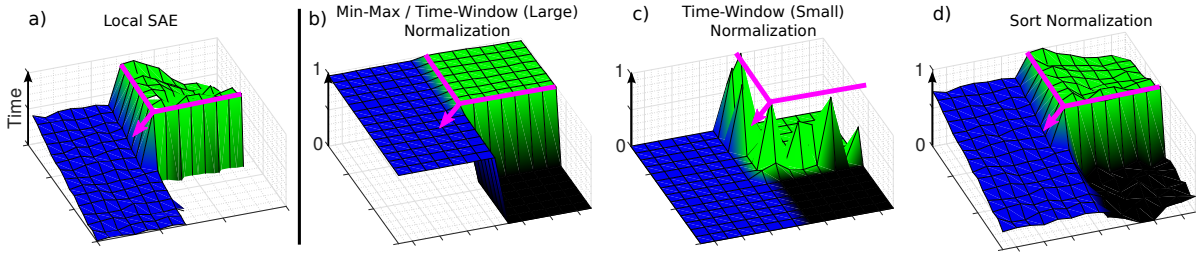

Figure 3. Comparison of different normalization approaches to form a descriptor for a corner-event located in a 90° corner (edge in magenta) from the local SAE (a), during a local motion in the indicated direction (magenta arrow). The current and previous positions of the edge where the corner is located are in green, while in blue are the previous locations of a completely different edge captured earlier. The colorless area represents locations with older timestamps that are out of range. In (b), the min-max is completely dominated by the oldest locations in the local SAE. The same situation arises under a relatively large time-window. Conversely, in (c), a too small time-window underepresents the locations of the current edge. The proposed Sort Normalization, in (d), provides the best representation of the original local SAE.

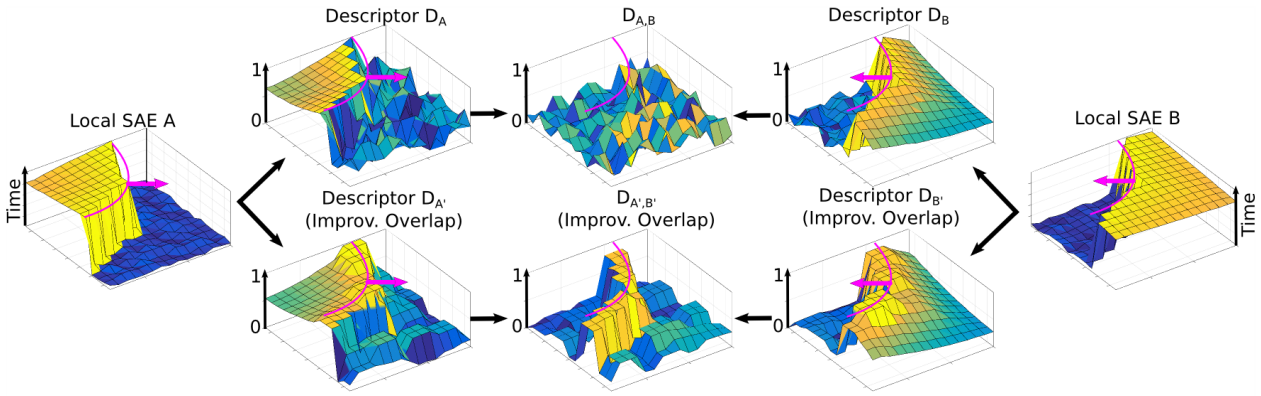

Figure 4. An edge (magenta) in which a tracked corner is located can induce a completely different local SAE depending on the direction of motion (magenta arrow). We compute the descriptors for two corner-event detections, A and B, from their respective local SAEs. The descriptors $D_{A'}$ and $D_{B'}$ or D_A and D_B indicate whether they make use of the improved overlap technique or not, respectively. The distance is inversely proportional to the sum of elements in the overlapping descriptor, and is smaller when using the proposed technique in $D_{A',B'}$ than without using it in $D_{A,B}$.

D_B, while penalizing those that are older in at least one of the descriptor patches (contributing to a larger d). Assuming the two descriptors captured at the same corner but two different time instants, the common locations with highest values shared by the descriptor represent the most probable location of the scene's edges.

In our experiments, we set the size of the descriptor patches to 15×15 pixels. Although the presented region descriptor is application-agnostic for event-based algorithms, in our pipeline we compute them online only on the detected corner-events. Note that the proposed descriptor is not a rotation-invariant representation. However, as the corner tracks are detected asynchronously, this descriptor is allows the matching of corners locally, performing even under high-speed rotations as reported in Section 6.

4.3. Overlap Improvement

The proposed distance metric measures the amount of overlap between patch descriptors, which is correlated to the resemblance of the local SAEs that generate them. Therefore, using the proposed descriptor to locally match consecutive corner-events of the same corner as described, we implicitly assume that they might be only small variations in the local SAE. This assumption does not hold when, for instance, a corner changes abruptly the direction of motion in the image plane as illustrated in Fig. 4.

To overcome this, each pixel location in the descriptor does not use the corresponding timestamp of the same pixel location in the local patch during the Sort Normalization. Instead, in order to improve the overlap, we use the maximum timestamp within a local neighbourhood of locations (3×3 pixels in this paper) in the SAE around each pixel location before applying Sort Normalization. This simple overlap improvement technique has been experimentally verified to improve the tracking performance, by allowing matching of consecutive corner-events from the same real corner that can abruptly change the direction of motion.

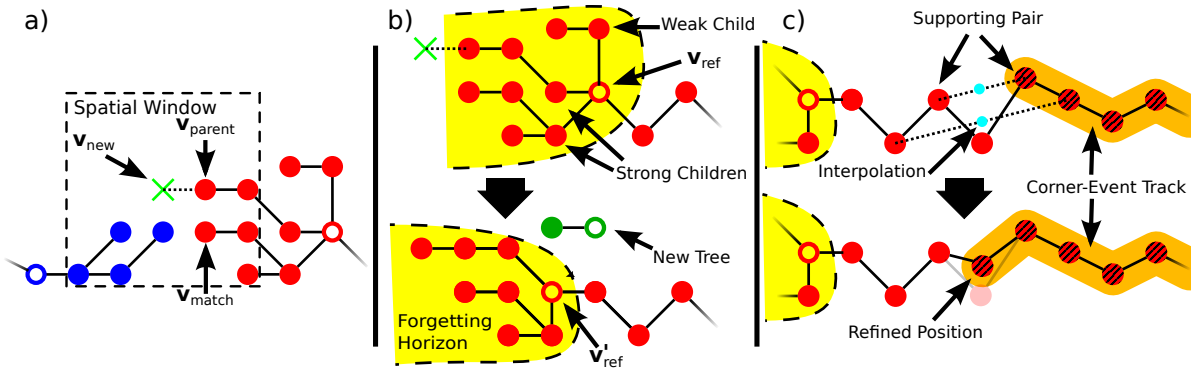

Figure 5. The stages of the ACE tracker: (a) tree assignment, (b) tree growth and updating of the forgetting horizon, and (c) the track refinement and retrieval. In (a), the descriptor of a new corner-event $\mathbf{v}_{new}$ is compared to vertices (circles) of two different trees, blue and red, and associated to the latter one. Thus, the forgetting horizon (here $\rho_{max} = 3$) of the red tree is updated by modifying $\mathbf{v}_{ref}$ as in (b). When $\mathbf{v}'_{ref}$ becomes the new reference vertex, the remaining strong child set it as its parent, whereas the other weak child initializes a new tree. In (c), the corner-event track (stripped circles) is updated by including the next vertex in the branch. Before reporting it as an asynchronous update of the track, the vertex gets its position refined based on the interpolation from the supporting pairs ($n_{smooth} = 2$ in this case).

5. Asynchronous Corner-Event (ACE) Tracker

The proposed algorithm employs an underlying directed graph representation, in which each vertex $\mathbf{v}$ is identified with an individual corner-event and encodes the same information $\mathbf{v} = \{t, x, y\}$ (i.e. the event's polarity is not used). The edges in the graph represent correspondence between detected corner-events.

Internally, the underlying graph is structured in smaller tree graphs T. Each of these trees accounts for a set of multiple hypotheses of trajectories for the same single corner in the scene defined in spatio-temporal space. The ACE algorithm describes how to associate new corner-events to the best tree, i.e. corner track, addressing the growth of these trees. As an individual tree grows, the different hypotheses for the corner track collapse into the most reliable one. The ACE algorithm is able to process new corner-event detections asynchronously and retrieve, also asynchronously, corner tracks. Following sections explain the different steps in the algorithm, illustrated in Fig. 5.

5.1. Tree Assignment

Upon the detection of a new corner-event, a corresponding vertex $\mathbf{v}_{new}$ is generated. The tracker attempts to associate $\mathbf{v}_{new}$ to other candidate vertices in neighbouring locations of the image array within spatial window. Each vertex in the graph can be 'active' (by default) or 'inactive', indicating whether it can be associated to new vertices or not, respectively.

Assuming perfect corner-event detection, would be sufficient to include only the immediate neighbours in the spatial window. However, to cope with misdetections, we employ 5×5 pixels window. From all the active vertices in the spatial window, we select the one with the closest distance when comparing descriptors with the new event (See Eq. (2)), namely $\mathbf{v}_{match}$, that belongs to the tree T_{match}. From all the vertices in the spatial window belonging to T_{match}, we select the newest one as $\mathbf{v}_{parent}$. We establish correspondence, adding a new edge and growing T_{match}, from $\mathbf{v}_{parent}$ to $\mathbf{v}_{new}$. However, if the descriptor distance between $\mathbf{v}_{new}$ and $\mathbf{v}_{match}$ is larger than a threshold d_{max} or there are no other active vertices in the spatial window, $\mathbf{v}_{new}$ becomes the root of a new tree instead. Note that we set $d_{max} = 0.5$, a relatively large matching margin to favour the growth of the trees in this step.

The inclusion of $\mathbf{v}_{new}$ in the graph deactivates any previous vertex in the same pixel coordinates, preventing any further data association with them. Additionally, any vertex within the spatial window older than Δt_{max} with respect to $\mathbf{v}_{new}$ also become inactive. Δt_{max} effectively implements a time-window to prevent data association with older and noisy parts. However, we set it to a conservatively high constant value of $\Delta t_{max} = 0.5$s to not hinder the tracking of slower corners.

5.2. Tree Growth and Forgetting Horizon

Each tree accounts for a special vertex referred as $\mathbf{v}_{ref}$, which keeps a maximum distance of ρ_{max} intermediate vertices to the deepest active vertex in the same tree. If, as result of including a new vertex in the tree, the depth of the tree is increased, $\mathbf{v}_{ref}$ is immediately updated.

To update $\mathbf{v}_{ref}$, we classify its active children vertices into strong or weak, depending on whether the descriptor distance to them is smaller or larger than a threshold d_{min}, respectively. We select the newest of the strong children or best weak children (according to the descriptor distance) as the new reference vertex $\mathbf{v}'_{ref}$. The remaining strong children vertices set $\mathbf{v}'_{ref}$ as their new parent or predecessor

vertex in the tree. The remaining weak children vertices become completely disconnected and initialize their own tree, acting as a root. After this graph mutation, the former reference vertex $\mathbf{v}_{ref}$ becomes inactive. Note that beyond the $\mathbf{v}_{ref}$, the tree becomes a single branch until the root vertex with this procedure.

The parameter ρ_{max} (set to $\rho_{max} = 10$ vertices here), accounts for a forgetting horizon that keeps active only the most promising branches (i.e. the most promising corner tracks hypotheses) as the tree grows asynchronously. In combination with the restrictive threshold $d_{min} = 0.25$, the forgetting horizon compensates for the overgrowth of the trees described in the previous step.

5.3. Track Retrieval and Refinement

Beyond the forgetting horizon, the remaining vertices up until the root represent the estimated corner track of the tree, accurately represented in spatio-temporal space and reported asynchronously every time $\mathbf{v}_{ref}$ is updated. However, depending on sensitivity of the chosen corner-event detector, multiple corner-events can detected in neighbouring locations around the same corner, as current state-of-the-art algorithms do not implement non-maxima suppression schemes. Therefore, the resulting asynchronous corner track might be jittery in the image plane.

In order to compensate for such effects, we introduce a simple and efficient algorithm to smooth the asynchronous corner track. When a single vertex is to be reported as an asynchronous update of the corner track, we first select the immediate i-th predecessor and successor vertices referred as a supporting pair, and repeat the process until the collection of n_{smooth} pairs. Between the vertices of the supporting pairs, an interpolated position at the time of the vertex to be reported is computed. The position of the vertex is refined by averaging it with the interpolated positions from the supporting pairs, and then reported for further processing. In this paper, we empirically set $n_{smooth} = 10$ pairs. Additionally, we only report the asynchronous corner-event tracks that contain at least 100 refined position vertices to prune the small and noisy tracks.

6. Experiments

We evaluate the proposed corner-event tracker on the publicly available event-camera dataset [14], which was recorded with a DAVIS sensor [3] with a resolution of 240×180 pixels capturing both event-based cues and intensity (frame-based) images using the same chip. As the proposed approach operates directly on the event-stream, the intensity images are used only for illustration. We test our algorithm on a representative subset of the dataset's sequences, namely on sequences from `shapes`, `poster` and `boxes` posing increasing levels of complexity, texture and average event-rate, often exhibiting fast camera motion causing significant blur in the intensity images. All of these sequences were recorded with a handheld sensor moving in a static scene and include IMU measurements as well as ground-truth camera poses.

We compare the proposed ACE tracker against the state-of-the-art event-based tracker of [23], referred to as 'EOF' (Event-based Optical-Flow tracker). We use the authors' public implementation of the method[1], and tune its parameters according to [22], where the method was evaluated on the same dataset. As in [23], we employ the version of the method that works on event cues only (and not with IMU cues) for fairness of comparison to our method.

The EOF tracker bootstraps features by detecting Harris corners in frames rendered from accumulated past events, once the optical flow of the scene has been corrected for. On these corner locations, a patch is initialized and tracked using Expectation-Maximization. These features are tracked at discrete time intervals spaced according to the estimated optical flow of the scene. If in any of the tracking iterations the number of features drops significantly, the method re-detects new features, attempting to maintain a constant number (set to 100 features in this paper as in [22]).

As opposed to the corner-patches of EOF, the proposed ACE tracker establishes correspondences only across corner-event features and tracks them. All the other events are still processed in order to compute the required descriptors for these corner-events. Moreover, in contrast to the EOF tracker, ACE adapts to the number of features in the scene, which is subject to the output of the corner-event detector employed. For this evaluation, we use the corner-event detector of [20] (including the improvements from [12][2]). Note that the proposed method implements a completely asynchronous tracking functionality in contrast to the EOF tracker, offering interesting advantages that are discussed in detail in the rest of this section.

6.1. Feature Tracks Evaluation

To compare the EOF to the proposed ACE tracker, we use the ground-truth camera poses available in each experiment and measure the quality of all the feature-tracks individually, by triangulating each feature in 3D using the known camera poses. If the resulting mean reprojection error for an individual feature-track is above 5 pixels, the feature track is considered invalid, or valid otherwise and thus deriving from it other performance metrics recorded in Table 1. Employing this methodology, we can reliably measure the quality of the tracking system without needing a complex SLAM back-end as in [22] or [17], and without relying on intensity frame-based images as in [23]. Figures 1 and 6 illustrate examples of the ACE tracker in action.

Table 1 records the valid tracks of event features (i.e.

[1] github.com/daniilidis-group/event_feature_tracking
[2] github.com/uzh-rpg/rpg_corner_events

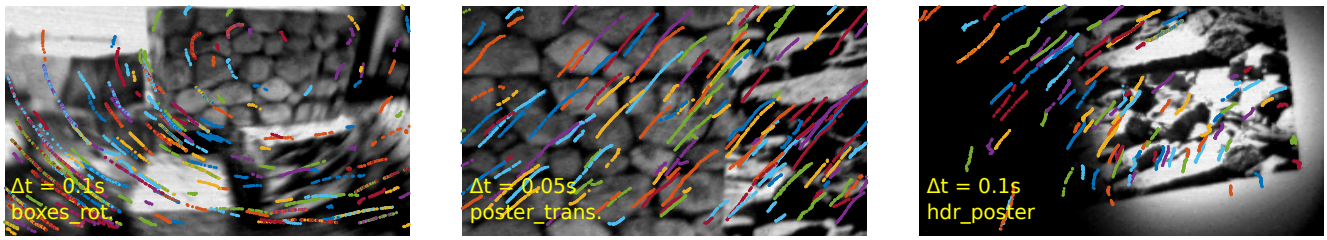

Figure 6. Examples of the method applied to different example scenes, during high-speed motions. The depicted asynchronous events are depicted as dots (color indicates association) over the last Δt time-window.

	Valid tracks [%]		Mean reproj. error [px]		Mean track-lifetime [s]		Mean no. tracked features [number/s]		Mean tracking frequency [kHz]	
Sequence \Tracker	EOF	ACE	EOF	ACE	EOF	ACE	EOF	ACE	EOF	ACE
shapes_rot.	**56.3**	41.3	**2.23**	2.49	0.08	**0.29**	**427**	31	0.26	**7.92**
shapes_trans.	**86.8**	78.5	1.93	**1.63**	0.15	**0.35**	**481**	52	0.20	**7.07**
shapes_6dof.	**77.3**	76.6	2.14	**1.91**	0.20	**0.34**	**371**	60	0.15	**6.84**
poster_rot.	**53.9**	41.6	2.32	**2.02**	0.19	**0.20**	**250**	89	0.25	**26.15**
poster_trans.	68.1	**70.2**	2.35	**1.66**	**0.41**	0.31	**191**	112	0.15	**14.36**
poster_6dof.	**73.8**	55.4	2.44	**2.02**	**0.30**	0.26	**270**	98	0.19	**17.40**
boxes_rot.	31.2	**32.9**	2.34	**2.28**	0.07	**0.17**	**160**	58	0.30	**33.28**
boxes_trans.	**85.4**	59.4	1.89	**1.50**	**0.29**	0.25	**300**	76	0.18	**16.79**
boxes_6dof.	**85.2**	53.3	2.00	**1.64**	0.23	**0.24**	**374**	66	0.25	**22.98**
hdr_poster	**86.2**	70.4	2.26	**1.75**	**0.35**	0.26	**280**	83	0.14	**10.11**
hdr_boxes	**87.8**	60.2	2.03	**1.54**	**0.33**	0.28	**298**	48	0.17	**13.60**

Table 1. Evaluation of feature-tracks for the EOF tracker [23] and the proposed ACE tracker. The best performance across each experiment and metric is highlighted in bold.

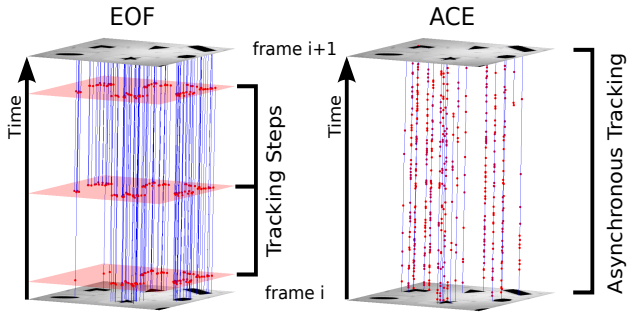

Figure 7. Example of the difference in the tracking modality of the ACE and EOF trackers, between two consecutive intensity frames (captured at 25Hz). The feature tracks (blue lines) are updated at specific time-instants (red dots) that can be either at discrete tracking steps (in the EOF tracker) or completely asynchronous (in the ACE tracker).

tracks with a mean reprojection error < 5px) as a percentage of all the detected tracks, as well as the mean reprojection error and mean lifetime of these valid tracks. As the EOF tracker does not impose a lower bound on the disparity of the detections of the feature tracks, it results to more valid tracks than ACE, in which we enforce minimum number of corner-events per track to be considered. Shorter tracks generally result in smaller reprojection errors (e.g. EOF tracker in `shapes_rot.`), as the feature detections are located closer to each other in the image plane. The resulting triangulated feature from such tracks, however, is more uncertain in depth due to the small parallax.

Considering the valid tracks, ACE is more accurate than EOF in terms of the mean reprojection error, resulting in generally longer track-lifetimes (stable even during abrupt motion changes in `shapes_rot.` and `boxes_rot.`). The robustness of the EOF in terms of track-lifetime under smooth motions (`poster` and `boxes` scenes excluding the rotation-only) arise from considering significantly more information than ACE between tracking iterations, incurring in higher computational cost as discussed in Section 6.2. Note that the performance of the proposed ACE drops in experiments where mainly rotation motions are used (indicated by `rot.`) when compared to other experiments, as the proposed descriptor is not rotation-invariant. Future work will investigate the development of a rotation-invariant version of the proposed descriptor to overcome this.

Table 1 also reports the mean number of tracked features within a moving window of 1s. Considering that the EOF, by design, attempts to maintain 100 tracked features in each iteration it is evident that several of these 100 features become invalid within each second, requiring re-detection of

	Mean event-rate [Mev/s]	Mean corner-rate [Mev/s]	Time per corner [μs/ev]	Real-time factor [%]
shapes_rot.	0.38	0.03	14.4	268
shapes_trans.	0.29	0.02	14.2	316
shapes_6dof.	0.30	0.03	20.1	191
poster_rot.	2.82	0.19	31.9	17
poster_trans.	1.66	0.17	28.6	30
poster_6dof.	2.22	0.15	32.5	20
boxes_rot.	3.09	0.22	35.1	13
boxes_trans.	1.87	0.13	34.1	23
boxes_6dof.	2.21	0.15	36.0	19
hdr_poster	1.72	0.10	27.7	35
hdr_boxes	1.98	0.11	31.2	30

Table 2. Computational performance of the proposed ACE Tracker per experiment.

new ones. As the ACE tracker does not impose an arbitrary constant number of features to be tracked, this number varies largely according to the texture of the scene while adapting to the number visual corners determined by the employed corner-event detector.

Lastly, the clear advantage of asynchronous tracking with the ACE algorithm becomes evident when evaluating the mean tracking frequency, i.e the frequency of update of a single feature track, in comparison to the EOF, which employs discrete tracking steps. While the EOF tracker updates its tracks at higher frequency than a conventional frame-based camera (25-30Hz), the proposed ACE can report track updates even beyond the rate of the ground-truth poses (200Hz), resulting in a difference of up to two orders of magnitude among the compared methods. The asynchronous tracking capability of ACE is especially suitable in high-frequency tracking, illustrated for qualitative comparison with EOF in Fig. 7.

6.2. Computational Performance

Table 2 reports the detailed computational performance of the proposed ACE algorithm per experiment, with all tests running on a Xeon E3-1505M CPU with 2.80GHz, 16GB of RAM and using a single-threaded C++ implementation. The table includes the mean event-rate (related to the scene's texture and motion speed), the mean rate of detected corner-events, and the mean time spent to process each single corner-event by the proposed algorithm. For clarity, we report on the 'Real-time factor', indicating the overall time spent processing the corner-events of each experiment with respect to the experiment's duration (i.e. results over 100% indicate a performance above real-time). ACE is able to track features several times faster than real-time in scene with moderate texture (e.g. `shapes`), while achieving a competitive performance in more complex experiments compared to other state-of-the-art event-based trackers (e.g. [4, 5]).

The author's public implementation of the EOF tracker is only in MATLAB, running hundreds of times slower than real-time even for the simplest experiments employing 4 threads, rendering EOF's computational cost not directly comparable. For a more accurate comparison with refer to [22], where a version of EOF with IMU cues is reported to run in real-time using a C++ implementation employing 6 threads, only under "moderate optical flow" and if the number of tracked features is capped to 15 (here, 100 features are used as in the evaluation of [22]) suggesting that the proposed ACE method is significantly more efficient. Note that the ACE tracker currently uses a single thread, but could be straightforwardly extended to a multi-threaded processing exploiting the sparsity of the corner-event detections for real-time performance.

7. Conclusions and Future Work

This paper presents a simple yet efficient local region descriptor of corner-events, which is applicable to any event-based algorithm requiring from data association. Building onto this descriptor, we propose the ACE tracker, able to retrieve asynchronous corner-tracks at an unprecedented frequency running in real-time in scenes of moderate complexity and performing even under highly dynamic motions. An evaluation on benchmarking datasets against the state-of-the-art reveals a significant boost in performance of the proposed approach, both in accuracy and computational efficiency.

Demonstrating the capabilities of asynchronous corner tracking when employing event cameras, this work opens up new research directions in event-vision. Future work will aim to exploit the power of this method to reliable track features in the event-stream to estimate the camera motion and ultimately, to develop a fully event-based SLAM framework able to process the detected corner-event tracks efficiently, towards the new asynchronous vision paradigm.

Acknowledgements: This work was supported by the Swiss National Science Foundation (SNSF, Agreement no. PP00P2 157585) and ECs Horizon 2020 Programme under grant agreement n. 644128 (AEROWORKS).

References

[1] I. Alzugaray and M. Chli. Asynchronous corner detection and tracking for event cameras in real time. *IEEE Robotics and Automation Letters*, 3(4):3177–3184, Oct 2018. 2

[2] R. Benosman, C. Clercq, X. Lagorce, S. H. Ieng, and C. Bartolozzi. Event-based visual flow. *IEEE Transactions on Neural Networks and Learning Systems*, 25(2):407–417, Feb 2014. 2, 3

[3] C. Brandli, R. Berner, M. Yang, S. C. Liu, and T. Delbruck. A 240×180 130db 3 μs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, Oct 2014. 1, 6

[4] X. Clady, S.-H. Ieng, and R. Benosman. Asynchronous event-based corner detection and matching. *Neural Networks*, 66(Supplement C):91 – 106, 2015. 2, 8

[5] X. Clady, J.-M. Maro, S. Barré, and R. B. Benosman. A motion-based feature for event-based pattern recognition. *Frontiers in Neuroscience*, 10:594, 2017. 2, 8

[6] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference*, pages 23.1–23.6. Alvety Vision Club, 1988. doi:10.5244/C.2.23. 2

[7] H. Kim, S. Leutenegger, and A. J. Davison. Real-time 3d reconstruction and 6-dof tracking with an event camera. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 349–364. Springer, 2016. 2

[8] X. Lagorce, C. Meyer, S. H. Ieng, D. Filliat, and R. Benosman. Asynchronous event-based multikernel algorithm for high-speed visual features tracking. *IEEE Transactions on Neural Networks and Learning Systems*, 26(8):1710–1720, Aug 2015. 2

[9] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman. Hots: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 39(7):1346–1359, July 2017. 2

[10] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128×128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, Feb 2008. 1

[11] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc. 2

[12] E. Mueggler, C. Bartolozzi, and D. Scaramuzza. Fast event-based corner detection. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2017. 2, 6

[13] E. Mueggler, C. Forster, N. Baumli, G. Gallego, and D. Scaramuzza. Lifetime estimation of events from dynamic vision sensors. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4874–4881, May 2015. 2

[14] 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. *International Journal of Robotics Research (IJRR)*, 36(2):142–149, 2017. 1, 6

[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*, 46(1):259–275, 2011. 1

[16] H. Rebecq, T. Horstschaefer, G. Gallego, and D. Scaramuzza. EVO: A geometric approach to event-based 6-DOF parallel tracking and mapping in real time. *IEEE Robotics and Automation Letters*, 2(2):593–600, 2017. 2

[17] H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2017. 2, 6

[18] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. 2

[19] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman. HATS: histograms of averaged time surfaces for robust event-based object classification. *CoRR*, abs/1803.07913, 2018. 2

[20] V. Vasco, A. Glover, and C. Bartolozzi. Fast event-based harris corner detection exploiting the advantages of event-driven cameras. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 4144–4149, Oct 2016. 2, 6

[21] D. Weikersdorfer, R. Hoffmann, and J. Conradt. Simultaneous localization and mapping for event-based vision systems. In M. Chen, B. Leibe, and B. Neumann, editors, *Computer Vision Systems*, pages 133–142, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 2

[22] A. Zhu, N. Atanasov, and K. Daniilidis. Event-based visual inertial odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2, 6, 8

[23] A. Z. Zhu, N. Atanasov, and K. Daniilidis. Event-based feature tracking with probabilistic data association. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4465–4470, May 2017. 2, 6, 7