



# Asynchronous Multi-Hypothesis Tracking of Features with Event Cameras

**Conference Paper****Author(s):**

Alzugaray Lopez, Ignacio ; Chli, Margarita 

**Publication date:**

2019-10-31

**Permanent link:**

<https://doi.org/10.3929/ethz-b-000360434>

**Rights / license:**

In Copyright - Non-Commercial Use Permitted

**Originally published in:**

<https://doi.org/10.1109/3DV.2019.00038>

**Funding acknowledgement:**

183720 - Collaborative Vision-based Perception, Towards Intelligent Robotic Teams (SNF)

# Asynchronous Multi-Hypothesis Tracking of Features with Event Cameras

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

[www.v4rl.ethz.ch](http://www.v4rl.ethz.ch)

## Abstract

With the emergence of event cameras, increasing research effort has been focusing on processing the asynchronous stream of events. With each event encoding a discrete intensity change at a particular pixel, uniquely time-stamped with high accuracy, this sensing information is so fundamentally different to the data provided by traditional frame-based cameras that most of the well-established vision algorithms are not applicable. Inspired by the need of effective event-based tracking, this paper addresses the tracking of generic patch features relying solely on events, while exploiting their asynchronicity and high-temporal resolution. The proposed approach outperforms the state-of-the-art in event-based feature tracking on well-established event camera datasets, retrieving longer and more accurate feature tracks at higher a frequency. Considering tracking as an optimization problem of matching the current view to a feature template, the proposed method implements a simple and efficient technique that only requires the evaluation of a discrete set of tracking hypotheses.

**Video** – [https://youtu.be/eguV\\_AIbteU](https://youtu.be/eguV_AIbteU)

## 1. Introduction

Event cameras [15, 21, 3] are bio-inspired visual sensors gaining popularity in Computer Vision and Robotics. Individual pixels of an event camera react independently and asynchronously to intensity changes, generating a so-called ‘event’ when the change is beyond a predefined threshold. A single event  $e = \{t, x, y, p\}$  is defined by the  $x$  and  $y$  pixel location and a unique timestamp  $t$  registering where and when the intensity changed in the image array, as well as the binary polarity  $p$  indicating whether the intensity has increased or decreased. Such events get accurately timestamped with microsecond resolution and are mostly unaffected by motion blur, rendering them particularly appealing in scenarios with fast dynamics, such as unmanned aircraft navigation and tracking of head-mounted displays. Since each pixel is able to generate events independently, these cameras are able to perform even in scenes with High

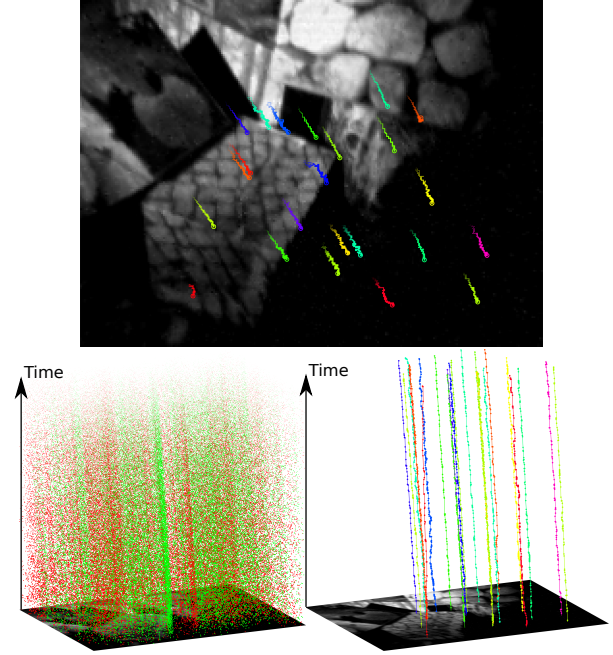


Figure 1: Set of tracked features in a HDR environment (boxes\_HDR from [20]), displaying their traces over the last 40ms in the image space (top) and superimposed to the corresponding gray-scale image (not used in our algorithm). Our method operates directly in the event stream (bottom left, color encodes polarity), that accumulate for such short time-interval over 100k events, and is able to retrieve high quality tracks with high temporal resolution (bottom right).

Dynamic Range (HDR) of illumination (e.g. see Figure 1). In this way, event cameras produce an asynchronous stream of events that encode all the visual experience of the camera as discretized and incremental intensity changes. This event stream is inherently very different from the concept of frames in traditional vision, where the pixels of a image frame (or image line in a rolling shutter camera) are read at the same time instant and register absolute intensity values. Such drastic differences between frame- and event-based sensing modalities renders, in most of the cases, the application of well-established algorithms to the event stream

largely impossible. As a result, this encourages the development of novel approaches aiming to exploit the power of the asynchronous event-vision paradigm.

In this paper, we address feature tracking for event cameras, which is a key component of most visual Simultaneous Localization And Mapping (SLAM) and Visual Odometry (VO) pipelines. Our approach solely relies on events and exploits the fine time-discretization provided by each event, assuming that no information is lost due to fast dynamics when employing event cameras (whereas traditional frame-based sensors are likely to experience motion-blur). Exploring this idea, here we propose a generic approach that is able to transform an underlying optimization-based formulation for feature tracking (e.g. the KLT tracker [16] for frame-based cameras), into a simpler and more efficient evaluation of likely tracking hypotheses. We believe that such an approach may find applications to other event-based problems (e.g. camera pose estimation) and thus, we first describe it in generic terms in Section 3.1. Based on such a formulation, we propose a powerful method to track templates of features described in Section 3.2, which operates asynchronously on an event-by-event basis. Our evaluation demonstrates better performance than the state-of-the-art in well-established datasets with realistic scenes [20], retrieving longer and more accurate feature tracks at a significantly higher frequency rate.

## 2. Related work

Feature tracking is one of the core components of visual SLAM [22] and VO [14], and several methods have been developed over the past decades for frame-based cameras. As a result, they are, in most of the cases, not able to cope or exploit asynchronous data streams, such as the event stream. In the early years of development of event cameras, research focused on tracking of known patterns in simplified scenarios. For instance, Censi *et al.* [5] tracked a rigid structure detecting the position high-frequency blinking LEDs. Tracking a known simple planar shape, Muegler *et al.* [19] achieved 6-Degrees-of-Freedom (DoF) camera pose tracking during aggressive motion, while Glover *et al.* [10] were able to detect and track a moving, single ball-shape in the scene. Event vision research quickly converged to tracking more generic features as in the work of Lagorce *et al.* [13] that addressed the tracking of distinctive shapes using a set of predefined kernels.

Some approaches have proposed the use of both frame- and event-based vision, for instance in [12] and [25], which characterized the features detected on intensity frames to later track them using solely events. A more advanced approach exploiting the combination of both sensing modalities can be found in [9].

Relying solely on events, Zhu *et al.* [28], proposed to apply an Expectation-Maximization (EM) algorithm to op-

timize for the location of the events in the image plane according to an estimate of the local optical flow, usually referred to as motion-compensation of events. The motion-compensated events are used to define the templates of the tracked feature on-the-fly and employed to track them as new events are generated. This method was successfully extended to consider cues from an Inertial Measurement Unit (IMU) and integrated into a VO pipeline in [27]. In a similar fashion, Rebecq *et al.* [24] proposed a tracking method that first compensates for the motion of the events using IMU and generates image-like patches that are later tracked using traditional KLT [16].

Both [24] and [28] integrate information over a fixed number of events or within a time-window in between tracking iterations, effectively defining ‘frames’ from events and partially defying the natural asynchronicity of the event stream. There is, however, an emerging trend in the development of algorithms that operate on an event-by-event basis, *i.e.* without artificially accumulating events and processing them as soon as they get captured instead, for which we can distinguish the efforts in asynchronous feature detection of corners [6, 26, 18, 2] or lines [4]. Establishing correspondence between such feature detections is not trivial in generic cases using an event-by-event strategy. For instance, Cladys *et al.* [7] matched asynchronously detected corners based on motion models. Alzugaray and Chli [2], and later Manderscheid *et al.* in the evaluation of [17], established local correspondence of corners based on the proximity of the detections. Using a more complex strategy, the work in [1] retrieved feature tracks considering a set of multiple hypotheses per each asynchronously detected corner in the event stream.

In this paper, we propose a method able to track generic patch features employing solely events as in [28]. However, instead of artificially accumulating events between tracking iterations, we implement an event-by-event approach that is able to process each event asynchronously. The method is inspired by [1] in the way that we explore different hypotheses per feature, but does not require from the detection of corners events and is able to process the raw event stream.

## 3. Methodology

### 3.1. Asynchronous Hypotheses Evaluation

Let us describe a generic optimization problem using event-data of the non-linear function  $f$  that depends on a state  $\mathbf{x}(t) \in \mathcal{X}$  (e.g. pose of the camera, feature position) and a set of events  $\mathbf{z}(t)$ . At a given  $t$  instant, we aim to optimize for the state,  $\mathbf{x}^*(t)$ , that fits best the events generated up to that instant,  $\mathbf{z}(t)$ , according to the function  $f$ . Formally,

$$\mathbf{x}^*(t) = \arg \max_{\mathbf{x}(t) \in \mathcal{X}} f(\mathbf{x}(t), \mathbf{z}(t)) \quad (1)$$

Assume that a given instant  $t_k$  the optimal solution  $\mathbf{x}^*(t_k)$  is known, for instance, at initialization (*e.g.* the origin frame for a camera pose or the detected pixel position of a visual feature). New events that arrive after  $t_k$  would necessarily modify  $\mathbf{z}(t)$  and thus, potentially, the optimal solution. However, the amount of information encoded in a single event is, in most of the cases, negligible and so are the changes that it may induce in the optimum of  $f$ . Several event-based methods exploit this fact and aggregate multiple events (*e.g.* a fixed number of them [23, 8, 24] or within a time window [28]) before re-optimizing Eq. (1).

Eq. (1) is usually optimized with iterative algorithms that require a good initialization point, typically seeded from the previous optimum state. However, if a large number of events gets aggregated before re-optimizing, the previous optimum might be too far from the new optimum state and thus the optimization is likely to fail. Analogously, in traditional frame-based cameras, the KLT tracker [16] is likely to fail if a feature exhibits significant displacement between consecutive frames. Conversely, our approach aims to leverage the information encoded in each single event as soon as it is captured, following an event-by-event processing strategy.

Let us define a small connected set of states  $\mathcal{S} \subseteq \mathcal{X}$ , that encloses the previous optimum state  $\mathbf{x}^*(t_k) \in \mathcal{S}$ , and its boundary  $\partial\mathcal{S}$ . Assuming a fine enough time discretization of the visual experience to be encoded in the event stream (*i.e.* there is no missing data as in the kidnapped robot problem), we expect the optimum  $\mathbf{x}^*(t)$  of  $f$  to transition smoothly in  $\mathcal{X}$  as new events get generated. Let us also define a discrete set of states lying in the boundary  $\mathbf{x}_{\partial\mathcal{S}} \in \partial\mathcal{S}$  that, jointly with the previous optimum  $\mathbf{x}^*(t_k)$ , define the set of hypotheses  $\mathbf{x}_h \in \mathcal{H}(\mathbf{x}^*(t_k)) = \{\mathbf{x}^*(t_k), \mathbf{x}_{\partial\mathcal{S}}\}$ . Note that we explicitly indicate the set of hypotheses  $\mathcal{H}(\mathbf{x})$  as a function to emphasize that  $\mathcal{S}$ , and therefore  $\mathbf{x}_{\partial\mathcal{S}}$ , is defined to enclose a given state  $\mathbf{x}$  (the previous optimum  $\mathbf{x}^*(t_k)$  in this case). While the optimum  $\mathbf{x}^*(t)$  lies within  $\mathcal{S}$  we could approximate the optimization in Eq. (1) with the following expression,

$$\mathbf{x}^*(t_{k+1}) = \arg \max_{\mathbf{x}_h \in \mathcal{H}(\mathbf{x}^*(t_k))} s(\mathbf{x}_h) \quad (2)$$

where  $s(\mathbf{x}_h) = f(\mathbf{x}_h, \mathbf{z})$ , namely the hypothesis score function. Note that we drop  $\mathbf{z}$  from the notation as all the hypotheses will always consider all the captured events at the moment they are evaluated. In our approach, the evolution of the score per hypothesis is tracked over time as new events are generated. Initially, the previous optimum  $\mathbf{x}_k^*$  would be the best hypothesis (*i.e.* the null hypothesis) for a subsequent set of new events. However, as more events are captured, a different hypothesis will eventually score higher than the null one, becoming itself the new optimum state

$\mathbf{x}^*(t_{k+1})$ , spawning a new set of hypothesis  $\mathcal{H}(\mathbf{x}^*(t_{k+1}))$ . This procedure allows us to track the global optimum of  $f$  as new events get generated.

Note that the region  $\mathcal{S}$  is to be defined small enough according to the problem specification (*e.g.* in the order of centimeters for camera position estimation or pixels for feature tracking). Additionally, the hypotheses are required to be well distributed in the boundary  $\partial\mathcal{S}$  so that optimum  $\mathbf{x}^*(t)$  cannot cross it without influencing any of them. In Section 3.2, we apply these concepts of the asynchronous hypotheses evaluation to the particular case of feature tracking employing solely events.

### 3.2. Asynchronous Event Tracker

The state of a feature is defined by the tuple  $\mathbf{x}_f = \{x_f, y_f, \theta\} \in \mathbb{R}^3$ , indicating its pixel position  $\mathbf{p}_f = \{x_f, y_f\}$  and rotation  $\theta_f$  in the image plane. Given the state of the feature  $\mathbf{x}_f$  we define an  $n \times n$  neighborhood  $\mathcal{N}(\mathbf{p}_f)$  centered at the feature location  $\mathbf{p}_f$ . Associated to each feature we define a template patch  $T \in \mathbb{R}^{n \times n}$  that, centered at the feature position  $\mathbf{p}_f$  and aligned according to the orientation  $\theta_f$ , indicates how likely it is that events get generated in neighboring locations. Details regarding the initialization and refinement of  $T$  are discussed in Section 3.2.1.

Each feature also keeps track of the last events  $\mathcal{E}$  that have been generated in its neighborhood  $\mathcal{N}$ . Note  $\mathcal{N}$  defines an image region that moves with the instantaneous feature position. For practical reasons, we assume than only  $m$  events are enough to determine the state of the tracker at any moment, so  $\mathcal{E} = \{e_0, e_1, \dots, e_m\}$  is the ordered set that considers only the latest events ( $e_m$  indicating the oldest one).

As soon as a new event is detected in a neighboring pixel location  $\mathbf{p}_e$  of the feature  $\mathbf{x}_f$ , *i.e.*  $\mathbf{p}_e \in \mathcal{N}(\mathbf{p}_f)$ , it replaces the oldest event in  $\mathcal{E}$ , as in a fixed-size circular buffer. Each event in the window  $e_i \in \mathcal{E}$ , originally detected at the position  $\mathbf{p}_i$ , is remapped to a new position  $\mathbf{p}'_i$  relative to the feature state as follows,

$$\mathbf{p}'_i(\mathbf{x}_f) = R^T(\theta_f)\mathbf{p}_i - \mathbf{p}_f, \quad R(\theta) \in SO(2) \quad (3)$$

We evaluate how likely is the event  $e_i$  to be generated in such relative location  $\mathbf{p}'_i$  by sampling the value of the template  $T$  (which is centered and aligned to the feature state  $\mathbf{x}_f$ ) at such location, defining the score function  $s$ , as:

$$s(e_i, \mathbf{x}_f) = T(\mathbf{p}'_i(\mathbf{x}_f)) \quad (4)$$

In fact, we are not only interested in how likely an event is to be triggered in a particular location, but also how likely it is that all the events in  $\mathcal{E}$  fit the template  $T$  according to the current feature state  $\mathbf{x}_f$ . We assign a relative importance to each of the  $m$  events in  $\mathcal{E}$  depending on their order of arrival,

so that the  $i$ th latest event in  $\mathcal{E}$  is assigned a Gaussian-like weight  $w_i = \exp(-\frac{1}{2}((i - \frac{m}{2})/\frac{m}{6})^2)$ . The total score given a feature state  $\mathbf{x}_f$  is then computed as follows,

$$s(\mathbf{x}_f) = \sum_{i=0}^m w_i s(e_i, \mathbf{x}_f), \quad e_i \in \mathcal{E} \quad (5)$$

Note that Eq. (5) can be used to optimize for the best state as  $\mathbf{x}_f^* = \arg \max s(\mathbf{x})$  at any time based on the events in  $\mathcal{E}$ ; this approach would be equivalent to the formulation in Eq. (1), albeit prohibitively expensive to be applied each time a new event is detected.

To avoid full optimization of the expression in Eq. (5), we consider instead evaluating the score function Eq. (5) not only at the current feature state  $\mathbf{x}_f$ , but also at a set of *hypothetical* feature states  $\mathbf{x}_h \in \mathcal{H}(\mathbf{x}_f)$  defined by small perturbations  $\delta \mathbf{x}_h \in \delta \mathcal{X}_h$ , i.e.  $\mathbf{x}_h = \mathbf{x}_f + \delta \mathbf{x}_h$ . Given a set of perturbations  $\delta \mathcal{X}_h$ , we can define the set of hypothetical states  $\mathcal{H}(\mathbf{x}_f)$  as described in Section 3.1 (including the null hypothesis,  $\delta \mathbf{x}_h = \mathbf{0}$ , so that  $\mathbf{x}_h = \mathbf{x}_f$ ). In practical terms, for the feature tracking problem, we define the set of perturbations as  $\delta \mathcal{X}_h = \{\mathbf{0}, (\pm x_{th}, 0, 0), (0, \pm y_{th}, 0), (0, 0, \pm \theta_{th})\}$ , where  $x_{th}$ ,  $y_{th}$  and  $\theta_{th}$  are small thresholds. This defined set represents the simplest and minimum set of disentangled perturbations (i.e. each perturbation only disturbs the feature state in a single dimension), although more complex ones could be employed.

The tracking method, as summarized in Algorithm 1, starts a new iteration each time a new event is integrated into  $\mathcal{E}$ , forcing the re-computation of the score  $s(\mathbf{x}_h)$  for each hypothesis  $\mathbf{x}_h$ . Following the formulation of Eq. (2), as soon as the best scoring hypothesis is different than the null hypothesis,  $\mathbf{x}_h^* \neq \mathbf{x}_f$ , it becomes the new feature state,  $\mathbf{x}_f \leftarrow \mathbf{x}_h^*$ , consequently spawning a new set of hypotheses  $\mathcal{H}(\mathbf{x}_f)$ . Note that, due to the bell-like shape of the weighting scheme  $w_i$ , the score  $s$  is mostly dominated by the events in the middle of the event window  $\mathcal{E}$ . Therefore, we assume that the transition from one hypothesis to another happens at the same instant as the middle event  $e_{m/2} \in \mathcal{E}$ . The tracking of the feature is abandoned the moment it leaves the field of view or if the feature state is not updated within a small time-window (50ms in this paper), as we assume the camera is always in motion.

Intuitively, our method is able to perform and transition to a correct hypothesis each time due to the consensus of all the events in  $\mathcal{E}$  with respect to the template  $T$ . Such consensus can be maintained over a sequence of different discrete hypotheses (through the evaluation of their respective score) because, exploiting the highly discretized temporal resolution of the event stream, no feature would possibly experience sudden jumps in the image plane, nor in translation or in rotation.

---

#### Algorithm 1 Tracking with Asynchronous Hypotheses

---

**Input:** Feature state  $\mathbf{x}_f$ , event window  $\mathcal{E}$ , template  $T$ .

- 1: New event  $e$  is generated at location  $\mathbf{p}_e$
  - 2: **if** ( $\mathbf{p}_e \in \mathcal{N}(\mathbf{x}_f)$ ) **then**
  - 3:   Update  $\mathcal{E}$ : replace the oldest event with  $e$
  - 4:   **for** each hypothesis  $\mathbf{x}_h \in \mathcal{H}(\mathbf{x}_f)$  **do**
  - 5:     Evaluate score  $s(\mathbf{x}_h)$  with Eq. (5)
  - 6:   **end for**
  - 7:   Select best hypothesis:  $\mathbf{x}_h^* = \arg \max_{\mathbf{x}_h \in \mathcal{H}} s(\mathbf{x}_h)$
  - 8:   Update tracker state:  $\mathbf{x}_f \leftarrow \mathbf{x}_h^*$
  - 9:   Refine template  $T$  (Section 3.2.1)
  - 10: **end if**
- 

Section 4 reports state-of-the-art performance for this simple but effective algorithm, despite the coarse set of discrete hypotheses determined by the thresholds  $x_{th} = y_{th} = 0.5\text{px}$  and  $\theta_{th} = 4^\circ$ . We also acknowledge the importance of the parameter  $m$  that defines the size of the event window  $\mathcal{E}$ , defined proportionally to the patch area with the parameter  $\beta$ , i.e.  $m = \beta n^2$ , which should be tuned accordingly to the texture of the tracked patch. However, we have observed that our smooth weighting scheme allows the application of reasonable values of  $\beta$  in a variety of scenarios with very different texture levels, as we will demonstrate fixing the parameter  $\beta = 0.2$  for the patch size  $n = 25$  for all the experiments, without further fine tuning.

#### 3.2.1 Feature template

The use of templates is common in event-based feature tracking, and they usually represent high-gradient regions in a location of interest. They are most reliably obtained from traditional intensity images [12, 9] that, provided they are well-conditioned (e.g. no motion blur, good illumination), capture an accurate snapshot of the scene invariant to the camera motion in contrast to the event stream. However, such templates can also be computed directly from event data to avoid relying on the quality intensity image snapshot, as done for example using the motion-compensated events approach of [28].

We propose initializing template  $T$  with a simple strategy where, given an initial feature state  $\mathbf{x}_f$ , we gather the latest  $m/2$  events that got generated in the neighborhood  $\mathcal{N}(\mathbf{x}_f)$  and wait until the next  $m/2$  are collected to initialize event window  $\mathcal{E}$ . All the events in  $\mathcal{E}$  are projected to the template  $T$  using the current feature state as described in Eq. (3), creating a density map in which each event contributes according to its weight  $w_i$  to the corresponding projected location in the template.

In general, the initial template  $T$  would not accurately represent the high-gradient regions of the feature patch, as  $\beta$  is not finely adjusted to the texture of each scene. However,



as the tracking progresses, we refine the template  $T$  in an event-by-event fashion. As soon as a new event is included in  $\mathcal{E}$ , the event in the center of the event window  $e_{m/2}$  is projected to the template  $T$  according to the current feature state  $\mathbf{x}_f$ , updating the value of the corresponding location as  $T(\mathbf{p}'_{m/2}(\mathbf{x}_f)) \leftarrow T(\mathbf{p}'_{m/2}(\mathbf{x}_f)) + \alpha w_{m/2}$ , with  $\alpha = 0.1$ . In the future, we will explore different strategies to automatically set  $\beta$  per feature as the template is refined.

## 4. Experimental Evaluation

### 4.1. Dataset and State-of-the-Art

We evaluate our algorithm in the Event Camera Dataset [20], that includes several scenes recorded with a DAVIS camera [3], a  $240 \times 180$  pixel sensor able to capture both events and images, although we only use the latter for visualization. We select three, hand-held recorded, static indoor scenes; namely *shapes*, *poster* and *boxes*, posing increasing complexity. Each of these scenes is recorded under pure translational movements (*trans.*), pure rotations (*rot.*) or both (*6dof.*). Additionally, we include in our evaluation the scenes *poster* and *boxes* recorded under HDR illumination. All the experiments last up to one minute and are recorded with increasing speed of camera motions, up to the point that the image frames exhibit significant motion blur. In all the experiments the ground-truth 6-DoF pose of the camera is available, captured from an external positioning system.

We compare our tracker to the method proposed by Zhu *et al.* [28], which is also tracks patch features relying solely on raw events. We employ the authors' open-source implementation<sup>1</sup> using the default parameters, only adapting the patch size. Their method also implements a simple strategy to detect features and initialize their tracker based on detecting Harris corners [11] on image of integrated events. For the sake of fairness, we initialize our features on the same initial feature positions such that we have a one-to-one correspondence per track with each method.

### 4.2. Benchmarking of Event-based Trackers

Benchmarking event-based tracking algorithms is, in general, a particularly challenging task as ground-truth is only available in simulation. For this reason, some works such as [2] and [9] compare their event-based tracks against the ones retrieved by the KLT tracker [16] applied to image frames. However, under fast dynamics or poor illumination conditions (as for most part of the experiments selected for evaluation in this paper) the KLT tracker consistently fails. Other works, such as [7] attempt to manually label the correct tracks, but this approach does not scale well as the scenes become more complex. An alternative would be to benchmark different tracking algorithms inspecting how

they perform in a generic VO pipeline able to process event-data, *e.g.* [27]. However, to the best of our knowledge, there exists no open-source implementation of any event-based VO system at submission time.

In this work, we use all the registered pixel locations of the feature track to triangulate its 3D position using the known camera poses (which are available as ground-truth in the chosen dataset). We project the triangulated 3D feature in the image plane and compare its location to the position of the feature track at each instant, defining the error of each such observation. Note that this approach does not necessarily imply that the 3D location of the triangulated feature is accurate (*e.g.* it will not be accurate regardless of the quality of the tracker if there is no parallax), but will evaluate whether the feature track position evolves consistently with the motion of the camera. This metric necessarily relies on the quality of the ground-truth poses and their alignment in time with the event stream, which are synchronized in the chosen dataset. This evaluation strategy is inspired by the one proposed originally in [1], and also later used in the evaluation of the method from [17].

For a each track, we define the reprojection error as the mean of distance of the reprojected 3D point to its feature track observations in the image plane. The defined error should not be inspected in isolation since, in general, shorter tracks should incur in a comparatively smaller parallax and proportionally smaller error. Ideally, tracks would be long and accurate and thus, we also consider the feature lifetime in our analysis. Therefore, in contrast to [1], here we define the error as a function of the track lifetime (unless otherwise stated), *i.e.* given a specific lifetime value, we triangulate the corresponding 3D position considering all the feature track observations until the selected lifetime value to obtain the average reprojection error.

All feature trackers are prone to failure of some of the individual tracks. However, this does not prevent their application to VO and SLAM system, as such pipelines are usually able to remove erroneous tracks if they do not match the general consensus. In our analysis, we distinguish between the tracks that have a reprojection error at a given lifetime below a threshold  $r_{th} = 5\text{px}$  (as in [1] and [17]), and consider them as inliers. The reported metrics only consider the inlier tracks for all the methods unless otherwise stated.

### 4.3. Quality of the Feature Tracks

Figure 3 depicts the mean error of the inlier tracks as a function of their lifetime per experiment, as well as a shaded area indicating the number of surviving inliers. In general, our algorithm performs comparably to the baseline method of [28] in terms of mean reprojection error of the inlier feature tracks in most of the experiments.

It is important to note that this reprojection error analysis

<sup>1</sup>[github.com/daniilidis-group/event\\_feature\\_tracking](https://github.com/daniilidis-group/event_feature_tracking)

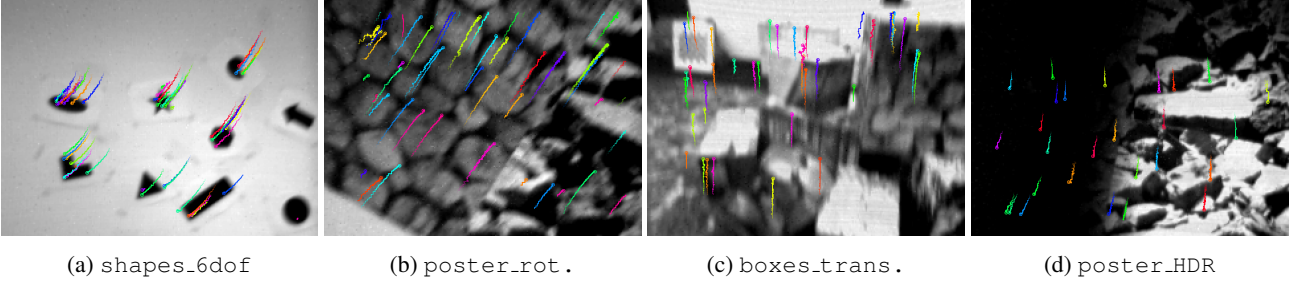


Figure 2: Example of event-based tracks asynchronously retrieved with the proposed method in different scenes of the dataset [20]. Each presented feature (color encodes different features) show its track aggregating the last 40ms, superimposed on the corresponding gray-scale image (not used in our algorithm). Note that the selected snapshots are purposely taken during high-speed camera motion, when the gray-scale images exhibit significant motion blur.

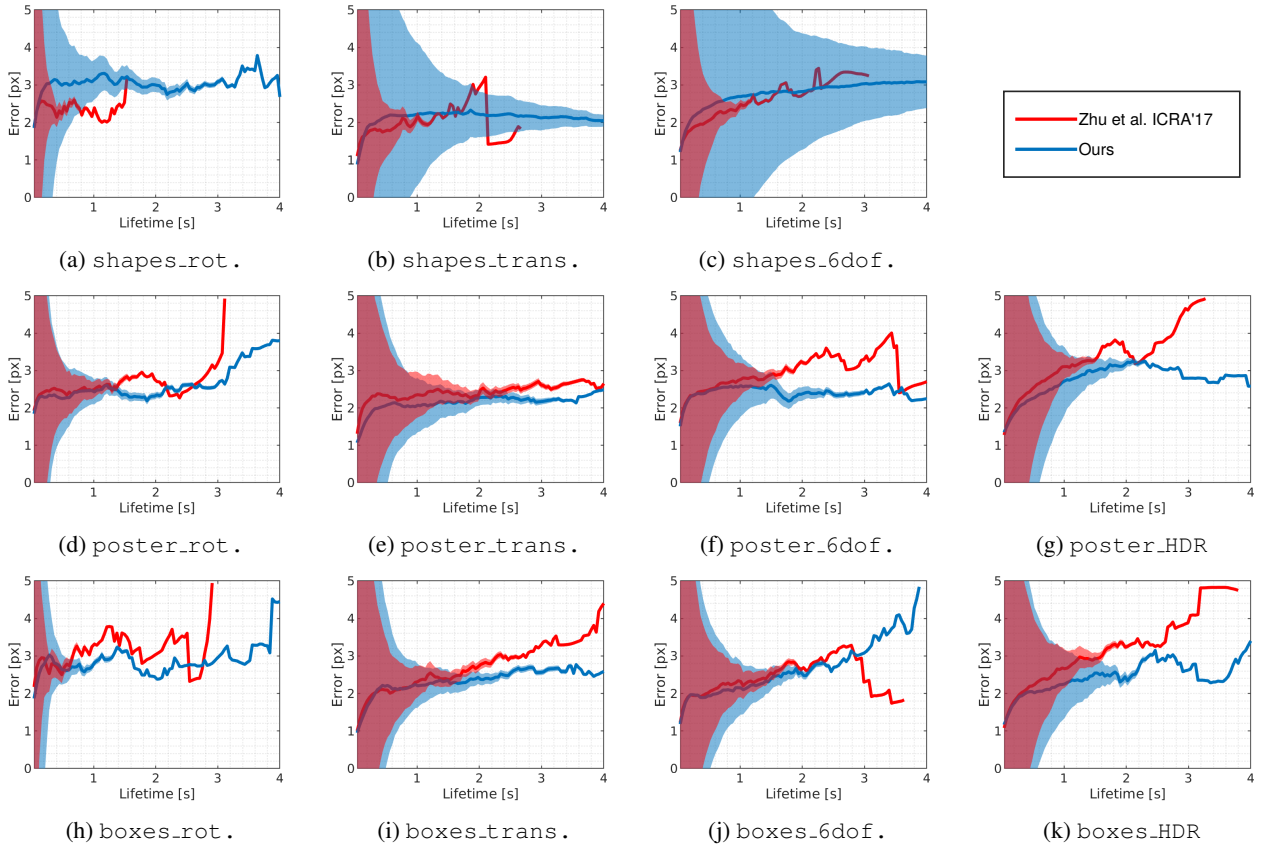


Figure 3: Rows indicate the scene and columns the motion or HDR illumination. The solid lines indicate the mean error of the inlier tracks for each method and the width of the corresponding shading area is proportional to the number of lasting inliers as a function of their lifetime. Note that, in most experiments, most of the inliers have a smaller lifetime than 3s.

is only valid when considering features with a short track-length. When considering longer track-lengths, we observe that our method performs equally or better than the baseline in most scenarios. For a clearer insight, we refer the reader to Figure 4, where the number of inlier tracks is depicted relatively to their lifetime. Although both methods exhibit a quick decay of the number of inlier tracks, in our method

the number of long lifetime features is always better than for the baseline method. As a result, the mean lifetime of our inlier tracks is always greater than those of the baseline. In particular, for *shapes*, where our method in some cases underperforms in terms of error, we can observe a significant improvement in the number of long lasting feature tracks with respect to the baseline.

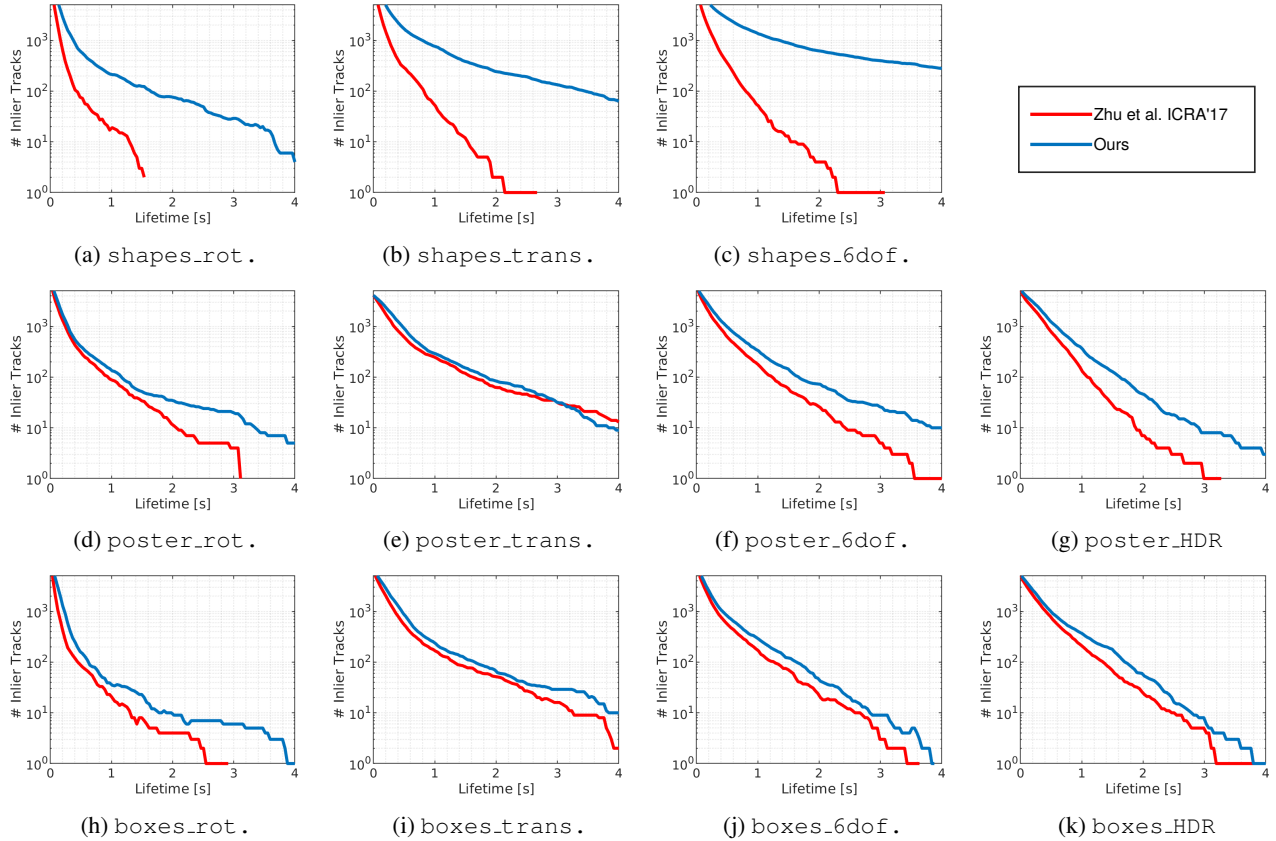


Figure 4: Rows indicate the scene and columns indicate the motion or HDR illumination. Each line represents the absolute number of inlier tracks as a function of their lifetime per method. Note that both method have the same number of initial features and they are initialized at the same pixel location.

Figure 3 and 4 illustrate quantitative results for both methods as when they would generate a set of completely independent tracks. Note, however, that in this evaluation our method is forced to employ the same feature detections as the baseline and thus direct one-to-one track correspondences can be established for both methods. For this reason, we also report the quality of each feature tracked with our algorithm for as long as the baseline method did (which is generally for a shorter time, as depicted in Figure 4); that is, setting the same lifetime value independently for each feature track. We present the results in Table 1 indicating the number of inlier tracks as a percentage over the initial number of tracks and their mean reprojection error under this constraint. We observe that, although the percentage of inliers is similar for both methods, given the same lifetime when comparing the same feature our method always obtains a lower reprojection error.

For qualitative examples of the feature tracks retrieved with the proposed method, we refer the reader to Figure 1 and Figure 2, where we have purposely chosen to illustrate

Experiment	Num. Inlier [%]		Error [px]	
	[28]	Ours	[28]	Ours
shapes_rot.	65	<b>70</b>	2.04	<b>1.84</b>
shapes_trans.	<b>90</b>	89	1.78	<b>1.10</b>
shapes_6dof.	82	<b>85</b>	1.97	<b>1.52</b>
poster_rot.	<b>55</b>	54	2.38	<b>2.11</b>
poster_trans.	71	<b>79</b>	2.41	<b>1.81</b>
poster_6dof.	<b>75</b>	69	2.43	<b>2.11</b>
poster_HDR	<b>84</b>	76	2.22	<b>1.79</b>
boxes_rot.	36	<b>51</b>	2.30	<b>2.09</b>
boxes_trans.	<b>84</b>	80	1.87	<b>1.39</b>
boxes_6dof.	<b>83</b>	76	1.98	<b>1.64</b>
boxes_HDR	<b>84</b>	73	2.06	<b>1.68</b>

Table 1: Mean reprojection error and percentage of inliers when tracking each feature up to the same lifetime with both methods.

the feature tracks captured under high-speed camera motion and when motion blur exists in the frame-based images.



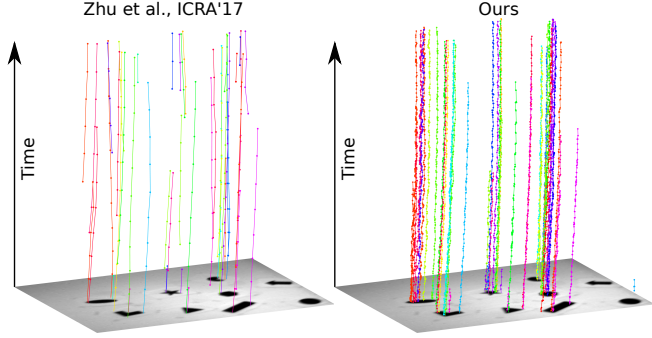


Figure 5: Same set of features tracked with the baseline [28] (left) and the proposed approach (right). Note the difference in the number of observations (dots) per each of the tracks

#### 4.4. High Frequency Tracking

One of the fundamental differences between the baseline method used here [28] and our approach is how the feature tracks get updated. The baseline method establishes different instants of time at which all the features are synchronously updated, which we refer to as tracking iterations. In each iteration, the next tracking iteration gets scheduled as a function of how fast the event stream is changing which, under high-speed camera motion, yields a higher rate of updates per feature track than we could possibly achieve with a regular frame-based camera.

However, imposing synchronous updates in the feature tracks defines artificial discretizations in the event stream, equivalent to conventional “frames”, essentially neglecting the asynchronous nature of the event cameras. Our approach fully exploits the asynchronicity of the events and allows each feature track to be updated as soon as new events are generated, following the event-by-event processing paradigm. Therefore, our approach is able to update asynchronously and independently each track at significant higher rate than the baseline method.

Figure 5 depicts a clear example (using the same scene and time interval as in Figure 3a) of the difference of the feature update rates for each method. We can observe that the baseline updates all the features at a given set of discrete times, retrieving only a small number of observations for each of them over time. Conversely, our method retrieves significantly more observations per track in the same time period. As a reference, the feature tracks are updated with our method at a mean rate of 13 kHz, whereas the baseline method only updates them at a mean rate of 145 Hz (averaging over all the experiments), *i.e.* more than an order of magnitude of difference.

#### 4.5. Computational Performance

Event cameras do not produce a fixed-rate output as conventional frame-based cameras, but instead they generate

events at different rates depending on the relative camera motion and the scene’s texture. Therefore, there is no general metric to indicate whether an algorithm would perform well in real-time in all scenarios. Instead, we analyze the maximum rate of events that an event-based algorithm can process per second, in order to determine its applicability to high-speed applications.

Our algorithm is able to process each event generated in the neighborhood of a feature track as fast as  $80\mu s$  per event, which is equivalent a maximum rate of 12500 events per second. This limit is computed using an unoptimized C++ single-threaded implementation, running in a Xeon E3-1505M CPU with 2.80GHz, 16GB of RAM, without GPU acceleration. In the selected experiments the rate of generated events spans from 0.3 to 3.1 millions of events per second, in `shapes.trans.` and `boxes.rot.`, respectively. Note, however, that such rates are a relative to the whole field of view, whereas our algorithm only processes events in the neighborhood of feature locations.

It is not possible to establish a fair comparison in terms of computational performance with the baseline method [28], as its open-source code is implemented in MATLAB. Such implementation may take up to hours to process a single experiment using CPU parallelization with 4 threads. For more insights on the baseline’s computational performance, we refer to the authors’ work [27], where a version of such algorithm using IMU cues is able to perform in real-time using a C++ implementation, but only under moderate optical flow and tracking only up to 15 features.

### 5. Conclusions

This paper presents an effective and novel event-based tracker for generic patch features that operates directly on the raw event stream, allowing its application under high-speed camera motion or poor illumination conditions. The method follows an event-by-event processing strategy and updates each of the feature tracks asynchronously and independently as soon as new events get generated. We developed the proposed algorithm based on a more generic multi-hypothesis framework that may find application to other event-based related problems. The presented evaluation reveals that the proposed method is able to retrieve, in general, longer and more accurate tracks than the state-of-the-art at a significantly higher frequency.

We believe that the proposed method opens up new research directions in the event-based community. In particular, future work will extend the proposed method to implement a full SLAM pipeline able to perform following the event-by-event paradigm.

**Acknowledgements:** This work was supported by the Swiss National Science Foundation (SNSF, Agreement no. PP00P2 183720).

## References

- [1] I. Alzugaray and M. Chli. ACE: An efficient asynchronous corner tracker for event cameras. In *International Conference on 3D Vision (3DV)*, Sep 2018. 2, 5
- [2] 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, 5
- [3] C. Brandli, R. Berner, M. Yang, S. C. Liu, and T. Delbruck. A  $240 \times 180$  130db 3  $\mu$ s latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, Oct 2014. 1, 5
- [4] C. Brandli, J. Strubel, S. Keller, D. Scaramuzza, and T. Delbruck. ELiSeD – an event-based line segment detector. In *International Conference on Event-Based Control, Communication and Signal Processing (EBCCSP)*, pages 1–7, June 2016. 2
- [5] A. Censi, J. Strubel, C. Brandli, T. Delbruck, and D. Scaramuzza. Low-latency localization by active led markers tracking using a dynamic vision sensor. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 891–898, Nov 2013. 2
- [6] X. Clady, S.-H. Ieng, and R. Benosman. Asynchronous event-based corner detection and matching. *Neural Networks*, 66:91–106, 2015. 2
- [7] 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, 5
- [8] G. Gallego and D. Scaramuzza. Accurate angular velocity estimation with an event camera. *IEEE Robotics and Automation Letters*, 2(2):632–639, April 2017. 3
- [9] D. Gehrig, H. Rebecq, G. Gallego, and D. Scaramuzza. Asynchronous, photometric feature tracking using events and frames. In *The European Conference on Computer Vision (ECCV)*, September 2018. 2, 4, 5
- [10] A. Glover and C. Bartolozzi. Event-driven ball detection and gaze fixation in clutter. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 2203–2208, Oct 2016. 2
- [11] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference*, pages 23.1–23.6. Alvey Vision Club, 1988. doi:10.5244/C.2.23. 5
- [12] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza. Low-latency visual odometry using event-based feature tracks. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 16–23, Oct 2016. 2, 4
- [13] 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
- [14] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. Keyframe-based visual-inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015. 2
- [15] P. Lichtsteiner, C. Posch, and T. Delbruck. A  $128 \times 128$  120 dB 15  $\mu$ s latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, Feb 2008. 1
- [16] 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, 3, 5
- [17] J. Manderscheid, A. Sironi, N. Bourdis, D. Migliore, and V. Lepetit. Speed invariant time surface for learning to detect corner points with event-based cameras. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 2, 5
- [18] E. Mueggler, C. Bartolozzi, and D. Scaramuzza. Fast event-based corner detection. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2017. 2
- [19] E. Mueggler, B. Huber, and D. Scaramuzza. Event-based, 6-dof pose tracking for high-speed maneuvers. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 2761–2768, Sept 2014. 2
- [20] 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, 2, 5, 6
- [21] 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
- [22] T. Qin, P. Li, and S. Shen. VINS-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, Aug 2018. 2
- [23] 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. 3
- [24] 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, 3
- [25] 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 (EBCCSP)*, pages 1–7, June 2016. 2
- [26] 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
- [27] 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, 5, 8
- [28] 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](#), [3](#), [4](#), [5](#),  
[7](#), [8](#)