Master project: Object tracking with event cameras

Viola Campos

Hochschule RheinMain, Wiesbaden

Abstract

Event-based vision promises to overcome some of the limitations of traditional frame-based vision. In high frequency tracking event cameras open up new possibilities as they do not suffer from motion blur during high-speed motions or saturation in scenes with high dynamic range. But to exploit these advantages, new algorithms are needed to process the unusual output of these cameras: an asynchronous stream of events. In this work we give an overview of current methods for event based tracking and implement a visual system which tracks moving objects using event based vision.

1. Introduction

Event cameras use a new approach to capture visual information which differs completely from traditionally frame-based computer vision. Inspired by the neural architecture of the eye, they capture changes in the surrounding scene asynchronously as per-pixel brightness changes, so called events. A single event is defined by its x and y pixel location in the image plane and a timestamp t indicating where and when the intensity changed, and a binary polarity p indicating whether the intensity has increased or decreased. Instead of measuring absolute brightness at a constant rate as in standard frame-based vision, they produce an asynchronous and sparse stream of events that encodes all the visual experience of the camera as discretized and incremental intensity changes. Figure 1 illustrates the principle.

Event-based sensors capture motion in the surroundings with a very high temporal resolution and low latency (both in the order of μs), are mostly unaffected by motion blur and are able to perform in scenes with high dynamic range of illumination (140 dB vs. 60 dB of standard cameras)[2].

Event-streams are inherently very different from the concept of frames in traditional vision, where all pixels of an image frame are read at the same time and register absolute intensity values. These differences render, in most of the cases, the application of well-established algorithms to the event-stream impossible. To exploit the advantages of event-based vision, complete novel methods are required.

In this work, we investigate different approaches for event-based object tracking. Since event cameras outperform classical cameras in challenging situations like high speed motion scenarios or high dynamic range scenes, event-based tracking should offer advantages especially in these situations. On the other hand does the processing of the event output pose several challenges. To

Email address: viola.campos@hs-rm.de (Viola Campos)

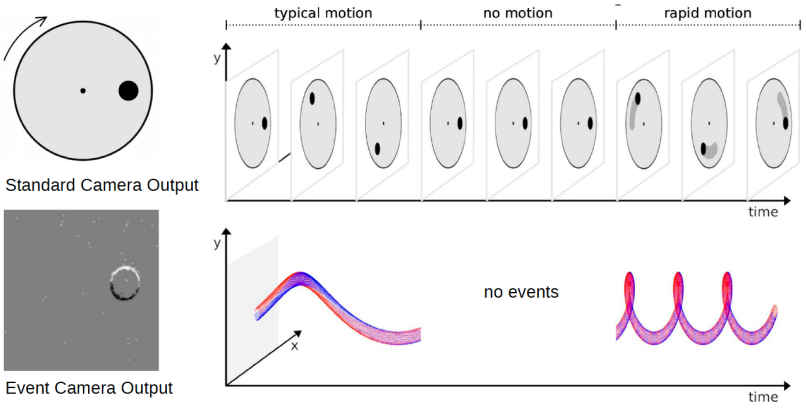

Figure 1: Output of a standard frame-based camera and an event camera when facing a black dot on a rotating disc [1]. Events are colored according to their polarity: blue means brightness increase, red means brightness decrease.

get a better grasp of the advantages and disadvantages of event-based tracking, we take a deeper look at two methods, which pursue different approaches: a work by Gehrig, Rebecq, Gallego and Scaramuzza from 2019 [3] which uses events together with additional greyscale frames for tracking and as a purely event-based approach, a corner tracker by Alzugarai and Chli from 2018 [4].

We developed a proof of concept implementation for both approaches and evaluated the proposed trackers on a publicly available event-camera dataset [5] and on live data from a DAVIS346.

2. Related Work

Tracking with event cameras is a major research topic, where the goal is to utilize the capabilities of event-based vision. A detailed survey of algorithms for event cameras can be found in Gallego et al. [2]. Algorithms for event-based object tracking can be divided into two groups: Methods operating on event data only and methods which use events together with additional information like greyscale frames or the output of an inertial measurement unit.

Representatives of the first group can be further divided into methods operating on a group of events and methods which process the stream event by event. Accumulating events during a short period allows to form artificial frame-like edge-images on which Repecq et al. [6] perform corner detection, inspired by frame-based FAST detection [7] and KLT tracking [8]. In [9], Zhu et al. accumulate events to artificial frames and compute correspondences between such frames with an Expectation Maximization (EM) approach. Both methods rely on the compensation of camera motion to obtain frame-like representations of the scene.

To fully exploit the asynchronous nature of the event-stream there exist several works which operate on a event-by-event basis. These systems are able to detect corners in the event-stream asynchronously as new events are generated. In [10], Vasco et al. propose an event-based corner detector inspired by the frame-based Harris detector [11], while [12] and [13] employ event-based FAST [7] corner detection. In [14] Alzugaray and Chli propose a simple tracker for such corner events based on proximity in the image space. [4] and [15] describe more mature tracking

schemes, introducing event-based local feature descriptors. Clady et al. [16] propose a corner-event tracker based on the estimated local optical flow.

Algorithms of the second category combine event data with additional data, usually with the output of a standard frame-based camera, to exploit the advantages of both systems. Kueng et al. [17] use greyscale frames to detect patches of Canny edges around Harris corners in grayscale frames and track such local edge patterns using Iterative Closest Point (ICP) on the corresponding event-stream. Earlier ICP based trackers combining events and frames are [18] and [19]. In [20] and [3] Gehrig et al. use grayscale frames together with a generative event model to estimate the optical flow that matches the observed frames to the event-stream.

3. Event-Based Tracking

Since event sensors capture visual information based on the scene dynamics, their potential lies in high speed motion scenarios which are challenging for frame-based cameras. Tracking fast moving objects with classical cameras, which is a classical problem in autonomous driving for example, is limited by the latency, the temporal discretization and the dynamic range of the camera [17]. Using event based vision, it should be possible to track objects even in high speed motion or challenging lighting situations.

On the other hand, event-based tracking introduces several challenges, which need to be addressed:

Data association. A main challenge in tracking objects with an event camera is the establishment of correspondences between events at different times. Frame-based trackers extract patches from the intensity frame as scene features which can be associated to similar patches in other frames. This cannot be transferred to event-based vision. As the sensor responds to temporal changes of intensity, the appearance of a feature varies depending on its motion, making data association difficult.

Data rate. Although event data is inherently sparse, as no redundant information is processed, the high temporal resolution can lead to high event-rates. Depending on the scene's texture and amount of motion in the scene, we experienced event rates up to 2 million events per second in our experiments with a DAVIS346. The maximum bandwidth of currently available event based cameras ranges from 1 million events per seconds (Meps) of a DAVIS128 up to 1200 Meps of a Samsung DVS-Gen4 [2].

Noise and dynamic effects. All vision sensors are noisy because of the inherent shot noise in photons and from transistor circuit noise. This situation is especially true for the relatively new event cameras, where the process of quantizing temporal contrast is complex and has not been completely characterized.

4. Methodology

4.1. Event Generation Model

Let $I(x, y, t)$ be the log-intensity value measured at a pixel location (x, y) where the last event was triggered at time $t - \Delta t$. A new event $e = (x, y, t, p)$ is generated as soon as the brightness increment ΔI at the pixel reaches a threshold $\pm C$ (with $C > 0$).

$$\Delta I(x, y) = I(x, y, t) - I(x, y, t - \Delta t) = pC, \tag{1}$$

where $p \in \{-1, 1\}$ denotes the event's polarity. Equation 1 is the event generation equation of an ideal sensor [21]. The contrast sensitivity $\pm C$ is a property of the event sensor used. Typical cameras can set thresholds between 10 %–50 % relative illumination change.

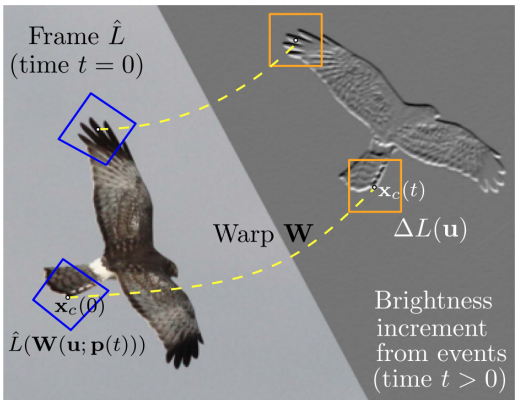

Figure 2: Illustration of tracking for two independent patches [3]. Events in a space–time window at time $t > t_0$ are collected into a patch of brightness increments (orange), which is compared, via a warp (i.e.,geometric transformation) $\mathbf{W}$ against a predicted brightness-increment image based on the frame at t_0 around the initial feature location (in blue).

4.2. Feature Tracking using Events and Frames

The first part of the project was an implementation of a feature tracker using events together with additional grayscale frames from [3]. A key component of the work are *brightness-increment images* which can be constructed either from events or from classical intensity frames. Accumulating event polarities pixel-wise over a time interval Δt produces an event-based image $\Delta I(x, y)$ with the amount of brightness change that occurred during the interval.

$$\Delta I(x, y) = \sum_{t_k \in \Delta t} p_k C \tag{2}$$

For small time slices Δt, the brightness increments in a small local region are caused by edges moving with a constant velocity. With this assumption the *optical flow equation* can be assumed to hold for all pixels within a small window. Hence for the grayscale intensity image, the optical flow constraint can be applied, which states that the gradient along time can be approximated as $\frac{\partial I}{\partial t} \approx -\nabla I \cdot \mathbf{v}$ where ∇I is the image gradient and $\mathbf{v}$ the motion velocity. The brightness change over a time interval Δt is thus

$$\Delta I(x, y) = \frac{\partial I}{\partial t} \Delta t \approx -\nabla I(x, y) \cdot \mathbf{v}(x, y) \Delta t \tag{3}$$

This equation is used as a forward model to predict brightness increments from the given intensity images. As a Using a maximum likelihood approach, the difference between the observed brightness changes from the events (2) and the predicted ones from the frames (3) is used as an error function to estimate the motion parameters that best explain the observed events. The brightness-increment images (2) and (3) are compared over small patches containing distinctive patterns. If these patches are extracted from an intensity frame at an initial time t_0 and compared to events in a space-time window at a later time Δt we need to model two types of motion: the

optical flow $\mathbf{v}$ during Δt which determines the appearance of the patch and the geometric transformation $\mathbf{W}$ which maps the initial patch location at t_0 to its location at Δt (see figure 2). The goal is to find warping parameters $\mathbf{p}$ and the velocity $\mathbf{v}$ that maximize the similarity between $\Delta I(x, y)$ given by events created during Δt and the predicted increment from a patch extracted from the greyscale frame at the warped possition $\mathbf{W}((x, y); \mathbf{p})$.

$$\Delta I((x, y); \mathbf{p}, \mathbf{v}) = -\nabla I(\mathbf{W}((x, y); \mathbf{p})) \cdot \mathbf{v}\Delta t \tag{4}$$

Figure 3 provides an overview of the computation of the brightness increments for a patch.

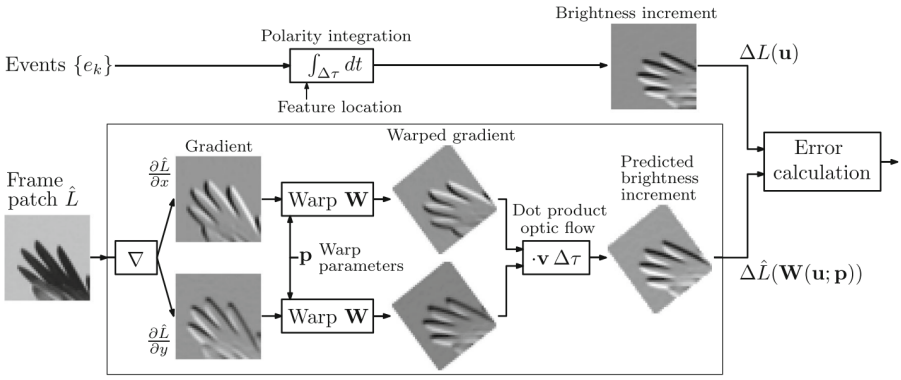

Figure 3: Block diagram showing how the brightness increments are computed for one of the patches in Figure 4. The top of the diagram depicts the brightness increment obtained by event integration (2), whereas the bottom of the diagram shows the generative event model stemming from the frame (3)

In our implementation we consider warps given by rigid-body motions in the image plane, consisting of a rotation and a translation and optimize the motion parameters by minimizing the squared L^2 norm of the difference between observed and modeled intensity increment. Other than indicated in equation (2) we do not collect events during a given period of time to form the event-based patches but a given number of events n_e. Collecting too many or too few events does not provide good representations of the underlying scene texture, so the optimal n_e per patch is constantly recomputed depending on the amout of structure in the patch. The objective function is optimized using a non-linear least squares framework. The steps of the feature tracker are

summarized in algorithm 1.

Algorithm 1: Photometric feature tracking using events and frames

Feature Initialization:
Detect Harris corners on the intensity frame $I(x, y)$, extract intensity patches around corner points and compute $\Delta I(x, y)$ for each frame-based patch.

Set event-based patches $\Delta I(x, y) = 0$, set initial warping parameters $\mathbf{p}$ to those of the identity warp, and set the number of events n_e to integrate on each patch.
Feature tracking:
for *each incoming event* **do**
    Update the patches containing the event (i.e., accumulate polarity pixel-wise)
    for *each patch* $\Delta I(x, y)$ *(once* n_e *events have been collected)* **do**
        Minimize the objective function, to get parameters $\mathbf{p}$ and optic flow $\mathbf{v}$
        Update the registration parameters $\mathbf{p}$ of the feature patch (e.g., position)
        Reset the patch $\Delta I(x, y) = 0$ and recompute n_e

4.3. Corner Event Tracking

Since experiments with the prototypical Python implementation showed that especially the optimization step is expensive, we investigated and implemented a second event-based tracking algorithm which uses a different approach. The tracker described in [4] detects corners in the event stream and defines local feature descriptors which can be used to establish local correspondence and to retrieve corner tracks more efficiently.

4.3.1. Event Cameras and Corner Events

Currently there exist two different approaches to asynchronously detect corners in event data. [10] describes Harris-based corner detection, while [12] and [13] employ event-based FAST corner detection. All works provide a reference implementation in C++. As our tracker is agnostic to the underlying corner-event detector, we were able to compare different corner-event detectors with only minor changes.

4.3.2. Local Region Descriptors

To establish correspondences between detected corner-events, we use a descriptor for the local spatio-temporal context of each corner-event. These descriptors are extracted from the *Surface of Active Events (SAE)* of the event stream. The SAE maps each pixel location in the image array to the timestamp of the most recent event triggered at this location, summarizing the event-streams current state. For each detected corner-event, a local patch surrounding the corner event is extracted from the SAE. Since these patches only contain absolute temporal information, they have to be normalized in order to make them comparable.

The distribution of the timestamps in the SAE is related to the way, event cameras perceive the visual scene: the areas with most recent timestamps are those where moving high contrast regions, i.e. edges are currently projected to the image plane while areas with older timestamps represent where these edges were before. To compute a normalized descriptor which retains the relative importance of the locations in the patch according to their timestamps, we employ Sort Normalization, sorting the timestamp in relative order and assigning each pixel position in the descriptor its relative order, normalized to be in range $[0, 1]$. As shown in [4], this provides

the best representation of the local SAE compared to other normalization techniques like time-window normalization or min-max normalization.

Similarity of descriptors is measured by the amount of overlap between them, the distance between two normalized descriptor patches D_A and D_B is

$$d(D_A, D_B) = 1 - \frac{\sum min(D_A, D_B)}{max(\sum D_A, \sum D_B)} \in [0, 1], \tag{5}$$

where $min(D_A, D_B)$ is an element-wise operation that results in the so-called overlapping descriptor $D_{A,B}$. Summing the values at each pixel position, we prefer common current descriptor timestamps (contributing to a smaller d), while positions whose timestamps are older in at least one of the descriptor patches are penalized. The divisor acts as a normalization term.

4.3.3. Asynchronous Corner Event Tracking

To track features over time, an underlying directed tree representation is used for each tracked feature, where vertices represent corner events and edges encode possible trajectories for corners in the scene.

For every newly detected corner-event, a corresponding vertex $\mathbf{v}_{new}$ is generated and the tracker tries to associate $\mathbf{v}_{new}$ to other candidate vertices in neighboring locations in the image space. Assuming perfect corner detection, it would be sufficient to consider only direct neighbors, but to cope with misdetections, all vertices in a spatial window with dimension d_{conn} (5 x 5 pixels in our experiments) are taken into account. Out of these candidates, we select the vertex $\mathbf{v}_{match}$ with the closest descriptor distance to the new event and out of all candidates belonging to the same tree as $\mathbf{v}_{match}$ we select the newest one as $\mathbf{v}_{parent}$, establishing correspondence by adding a new edge between $\mathbf{v}_{parent}$ and $\mathbf{v}_{new}$. However, if there are no other vertices in the spatial window or if the descriptor distance between $\mathbf{v}_{new}$ and $\mathbf{v}_{match}$ is greater than a threshold d_{max}, $\mathbf{v}_{new}$ becomes the root of a new tree instead. Additionally, every vertex in the spatial window older than Δt_{max} with respect to $\mathbf{v}_{new}$ becomes inactive, such that it can no longer be associated to new vertices, to prevent data association with older or noisy parts.

Each tree accounts for a set of possible trajectories for the same corner in the scene. As the tree grows, these different hypotheses for the corner track collapse into the most reliable one. To accomplish that, each tree contains a special vertex $\mathbf{v}_{ref}$, which keeps a maximum distance of p_{max} intermediate vertices to the deepest active vertex in the same tree. If including a new vertex in a tree increases the trees's depth, $\mathbf{v}_{ref}$ is immediately updated. To update $\mathbf{v}_{ref}$, its children are classified into strong and weak children, depending on whether their descriptor distance to $\mathbf{v}_{ref}$ is smaller or larger than a threshold d_{min}. The newest of the strong children or best weak children (according to descriptor distance) becomes the new reference vertex $\mathbf{v'}_{ref}$ while the former $\mathbf{v}_{ref}$ is set inactive. The remaining strong children set $\mathbf{v'}_{ref}$ as their new parent, the remaining weak children become completely disconnected and initialize new trees. With this procedure, the tree becomes a single branch between root and $\mathbf{v}_{ref}$ and branches only after $\mathbf{v}_{ref}$. The parameter p_{max} models a forgetting horizon, that keeps only the most promising corner track hypothesis and prevents overgrowth of the tree. Figure 4 illustrates the process.

A shortcoming of the tracker is, that depending on the sensitivity of the chosen corner-event detector and the size of the spatial window d_{conn}, multiple corner-events are detected in neighboring locations. As the described algorithm will aggregate all corner-events to tracks, the resulting tracks become jittery. To compensate for this effect, we smooth tracks with the following method. For each vertex $\mathbf{v}$ of the final track, we select so called supporting pairs consisting of $\mathbf{v}$'s immediate i-th predecessor and successor and compute the interpolated vertex position for each pair.

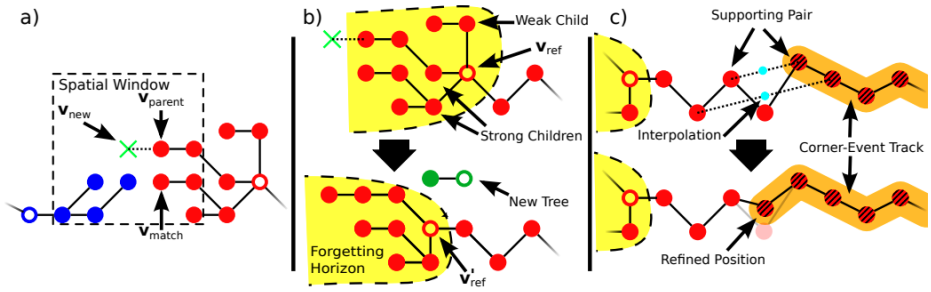

Figure 4: The stages of the ACE tracker: (a) tree assignment, (b) tree growth and updating of the forgetting horizon, and (c) track refinement and retrieval. In (a), the descriptor of a new corner-event $\mathbf{v}_{new}$ is compared to vertices (circles) of two different trees, blue and red, and associated to the latter one. Thus, the forgetting horizon (here $p_{max} = 3$) 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) [4].

A parameter n_{smooth} determines how many of these pairs get collected. The position of vertex $\mathbf{v}$ is then refined by averaging it with the interpolated positions from the supporting pairs.

5. Experiments

5.1. Feature Tracking using Events and Frames

We implemented the tracker as single threaded python application. To initialize the camera and transfer event data from the device, we use the DV application from inivation. The application allows to define event and image servers that provide the data as packets over a socket connection.

For the expensive minimization of the objective function, measuring the error between observed and predicted brightness increment, we used the C++ based non-linear least squares framework provided in the Ceres software [1], which was included into the project through a python wrapper. Figure 5 shows the output of the application.

The computational performance of the tracker was much worse than expected. On average, the tracker was able to process 500 events/s which means with a mean event rate of 800000 events/s (what we observed as a realistic value for indoor scenes) the processing time for a sequence was roughly 1800 times the original duration of the sequence.

One reason for the high processing time is the asynchronous nature of the algorithm. Incoming events are processed sequentially, where each event updates the state of several patches and the optimization step is triggered as soon as a patch has collected a certain number of patches, which makes the algorithm hardly parallelizable. The second problem is the expensive Expectation Maximization scheme that estimates the optical flow.

[1] http://ceres-solver.org/

Figure 5: Screenshot of the feature tracker. Circles mark currently tracked features, lines show the current optical flow for each feature

5.2. Corner Event Tracking

The purely event based corner tracker from [4] was implemented as single threaded C++ application, which can either use event data directly from the DAVIS346 device or read events from a dataset. The application supports the use of different corner detectors, we used a Harris- and a FAST-based detector in our experiments.

Depending on the event rate or rather the amount of texture and motion in the scene, the computational performance differed greatly. Table 1 shows the results of different experiments using datasets from [5] with increasing complexity. The event rate is the mean number of events processed per second, the corner rate indicates the percentage of events which were classified as corners and the real time factor gives the time spent to process the events of an experiment with respect to the experiment's duration.

sequence	detector	event-rate [ev/s]	corner-rate [%]	Real-time factor [%]
shapes_translation	fast	37900	4.71	27
shapes_translation	harris	21300	8.27	15.8
shapes_rotation	fast	46400	4.44	36.7
shapes_rotation	harris	19100	8.7	13.2
boxes_6dof	fast	6300	4.12	4.6
boxes_6dof	harris	13800	2.83	4.2

Table 1: Computational performance of the tracker in different scenes.

Figure 6 shows the quality of the tracks for simple scenes. Table 2 gives the mean track length and the mean number of tracked features.

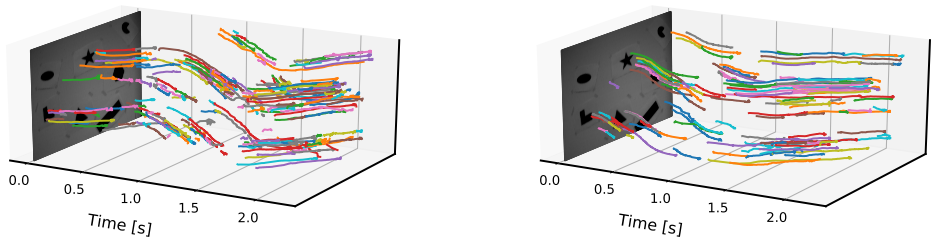

Figure 6: Event based feature tracks for `shapes_rotation` and `shapes_translation` from [5]. Greyscale frame is added for illustration.

sequence	corner detector	Mean track length [ev/s]	Mean # tracked features /s
shapes_translation	fast	17	133
shapes_translation	harris	29	117
shapes_rotation	fast	21	115
shapes_rotation	harris	38	134
boxes_6dof	fast	21	9684
boxes_6dof	harris	25	1098

Table 2: Quality of feature tracks in different scenes.

In our experiments, the performance of the tracker on live event data, directly transferred from a DAVIS346 was poor, compared to the simpler test cases from the event dataset. The corner detector did not reliably detect moving corners in the captured scene which led to short and noisy tracks. Figure 7 shows the problem. However, the tracker performed much better on real world scenes from the dataset. Figure 8 shows feature tracks for the `boxes_6dof` scene from [5].

The DAVIS346 offers the possibility to set various bias values individually. As this process comprises 25 different biases, each consisting of a coarse and fine value plus additional flags [2], the process of finding optimal biases for an application is not trivial. Based on recommendations from [5] we performed a rough grid search, but could not find settings, which significantly improved the results. Since the recommended settings from [5] are for the DAVIS240 and can't be directly transferred to the DAVIS346, we think a further, more thorough search for optimal parameters would lead to significant improvements.

6. Conclusions

In this work we studied and implemented two different algorithms for event based object tracking. As the experiments showed that the first algorithm [3] was way too complex for a

[2] see https://inivation.com/hardware4/biasing

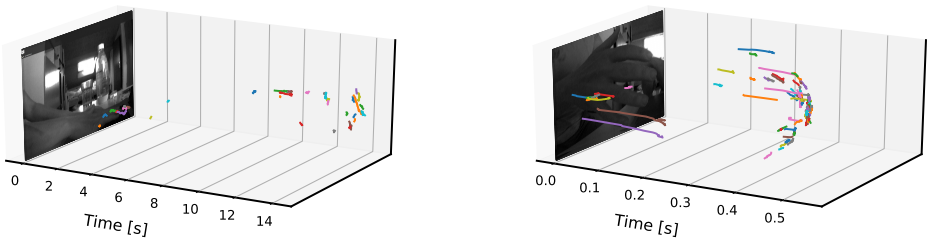

Figure 7: Event based feature tracks for live event data from the DAVIS346.

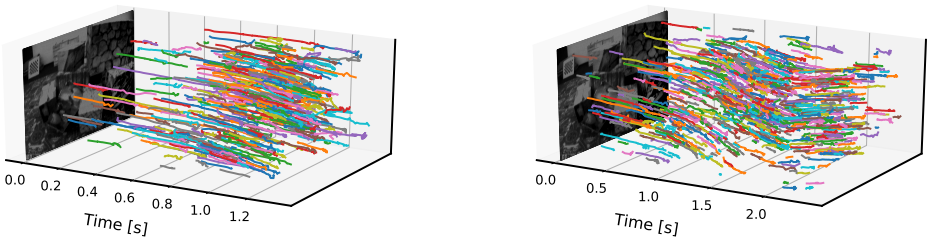

Figure 8: Feature tracks for `boxes_6dof` from [5]. Left figure shows tracks generated with FAST corner detector, right figure uses Harris detection.

proof-of-concept implementation in python, we focused on the implementation of the second algorithm [4]. The experiments showed that the quality of the tracking as well as the computational performance depend strongly on the complexity of the scene. While the tracker performs well in simple setups, tracking in more complex 'real life' scenes is difficult. Finding good bias settings for the camera also turned out to be unexpectedly complex.

References

[1] H. Kim, S. Leutenegger, A. Davison, Real-time 3d reconstruction and 6-dof tracking with an event camera, volume 9910, pp. 349–364.

[2] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, D. Scaramuzza, Event-based vision: A survey, ArXiv abs/1904.08405 (2019).

[3] D. Gehrig, H. Rebecq, G. Gallego, D. Scaramuzza, Eklt: Asynchronous photometric feature tracking using events and frames, International Journal of Computer Vision (2019) 1–18.

[4] I. Alzugaray, M. Chli, Ace: An efficient asynchronous corner tracker for event cameras, in: 2018 International Conference on 3D Vision (3DV), IEEE, pp. 653–661.

[5] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, D. Scaramuzza, The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam, The International Journal of Robotics Research 36 (2016).

[6] H. Rebecq, T. Horstschaefer, D. Scaramuzza, Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization, in: Proceedings of the British Machine Vision Conference (BMVC).
[7] E. Rosten, T. Drummond, Machine learning for high-speed corner detection, volume 3951.
[8] B. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision (ijcai), volume 81.
[9] A. Zhu, N. Atanasov, K. Daniilidis, Event-based feature tracking with probabilistic data association, pp. 4465–4470.
[10] V. Vasco, A. Glover, C. Bartolozzi, Fast event-based harris corner detection exploiting the advantages of event-driven cameras, pp. 4144–4149.
[11] C. Harris, M. Stephens, A combined corner and edge detector, Proceedings 4th Alvey Vision Conference 1988 (1988) 147–151.
[12] E. Mueggler, C. Bartolozzi, D. Scaramuzza, Fast event-based corner detection.
[13] R. Li, D. Shi, Y. Zhang, K. Li, R. Li, Fa-harris: A fast and asynchronous corner detector for event cameras.
[14] I. Alzugaray, M. Chli, Asynchronous corner detection and tracking for event cameras in real time, IEEE Robotics and Automation Letters 3 (2018) 3177–3184.
[15] I. Alzugaray, M. Chli, Asynchronous multi-hypothesis tracking of features with event cameras, in: 2019 International Conference on 3D Vision (3DV), pp. 269–278.
[16] X. Clady, S.-H. Ieng, R. Benosman, Asynchronous event-based corner detection and matching, Neural Networks in press (2015).
[17] B. Kueng, E. Mueggler, G. Gallego, D. Scaramuzza, Low-latency visual odometry using event-based feature tracks, pp. 16–23.
[18] Z. Ni, A. Bolopion, J. Agnus, R. Benosman, S. Régnier, Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics, Robotics, IEEE Transactions on 28 (2012) 1081–1089.
[19] Z. Ni, S.-H. Ieng, C. Posch, S. Régnier, R. Benosman, Visual tracking using neuromorphic asynchronous event-based cameras, Neural computation 27 (2015) 1–29.
[20] D. Gehrig, H. Rebecq, G. Gallego, D. Scaramuzza, Asynchronous, Photometric Feature Tracking Using Events and Frames.
[21] G. Gallego, C. Forster, E. Mueggler, D. Scaramuzza, Event-based camera pose tracking using a generative event model, CoRR abs/1510.01972 (2015).