ELiSeD – An Event-Based Line Segment Detector

Christian Brändli[1], Jonas Strubel[1], Susanne Keller[1], Davide Scaramuzza[2] Tobi Delbruck[1]
1: Institute of Neuroinformatics, University of Zurich, Winterthurerstrasse 190, Zurich, Switzerland.
2: Robotics and Perception Group, University of Zurich, Andreasstrasse 15m Zurich, Switzerland,

***Abstract*— Event-based temporal contrast vision sensors such as the Dynamic Vison Sensor (DVS) have advantages such as high dynamic range, low latency, and low power consumption. Instead of frames, these sensors produce a stream of events that encode discrete amounts of temporal contrast. Surfaces and objects with sufficient spatial contrast trigger events if they are moving relative to the sensor, which thus performs inherent edge detection. These sensors are well-suited for motion capture, but so far suitable event-based, low-level features that allow assigning events to spatial structures have been lacking. A general solution of the so-called event correspondence problem, i.e. inferring which events are caused by the motion of the same spatial feature, would allow applying these sensors in a multitude of tasks such as visual odometry or structure from motion. The proposed Event-based Line Segment Detector (ELiSeD) is a step towards solving this problem by parameterizing the event stream as a set of line segments. The event stream which is used to update these low-level features is continuous in time and has a high temporal resolution; this allows capturing even fast motions without the requirement to solve the conventional frame-to-frame motion correspondence problem. The ELiSeD feature detector and tracker runs in real-time on a laptop computer at image speeds of up to 1300 pix/s and can continuously track rotations of up to 720 deg/s. The algorithm is open-sourced in the jAER project.**

Keywords—event-based; computer vision; machine vision; line segment detector; visual feature; DVS; DAVIS; silicon retina

I. Introduction

Real-time interaction with the environment on mobile platforms such as mobile robots or mobile devices requires low reaction times on a small power budget. On actuated mobile platforms, such as drones, the control problem can be facilitated if the sensing delay is minimized and the control error can be measured instantaneously [1]. For an immersive experience of virtual or augmented reality a system latency of 20ms or below must be achieved [2]. With frame-based vision sensors, low latencies can be achieved by increasing the frame rate which in turn increases the power consumption. Achieving low latencies without increasing power consumption requires a more efficient way of encoding the visual information. This efficiency can be achieved by avoiding the processing of redundant data of pixels that do not change in between frames.

A novel type of vision sensor, called the Dynamic Vision Sensor (**DVS**) [3], performs such redundancy suppression by asynchronously communicating only the addresses of pixels where the change in the log light intensity exceeds an upper or lower threshold [4]. Thereby, the sensors achieve real-world sub millisecond latencies and an intra-scene dynamic range of 130 dB at an average power consumption of 10 mW [5]. Instead of sampling the visual information with a fixed frame rate, changes in the visual information are encoded by the pixels without the requirement of external signals. This way even fast motions are captured, which allows tracking objects continuously and circumventing the motion correspondence problem. This paper presents an algorithm to detect and track simple generic contours and shapes as a set of line segments that are extracted from the event stream without prior knowledge on the structures.

This research has been supported by the European Union funded project SeeBetter (FP7-ICT-2009-6), the Swiss National Science Foundation through the NCCR Robotics and the Gebert Rüf Foundation through GRS-048/14.

II. Event-Based Vision Sensors

The most common form of event-based vision sensors is based on the dynamic vision sensor pixel [3], [6]. The principle of these pixels is shown in Fig. 1a: The light intensity at the pixel is log compressed, asynchronously sampled and changes are amplified. As soon as a change exceeds an upper (θ_{ON}) or lower threshold (θ_{OFF}), the pixel is reset, the latest value sampled and it generates a so-called "ON event" if it gets brighter or an "OFF event" if it gets darker. The information transmitted with each event is its "polarity" (ON or OFF, i.e. the sign of the change), its address and the time of its creation (with microsecond resolution). The output of the sensor is therefore a continuous stream of such timestamped address-events which are usually communicated in event packets. Each event nominally represents a quantized change of log intensity, although sensor non-idealities can cause significant deviations from a uniform response. To render the output, the events in a given time interval are integrated for each pixel and its brightness is increased or decreased according to the polarity of the events as shown in Fig.1b.

The manufacturer of the first commercially available event-based vision sensor DVS128 (http://www.inilabs.com), released an improved sensor in 2014. This new sensor is called the Dynamic and Active Pixel Vision Sensor (**DAVIS**). The model DAVIS240 [5] camera used in this paper has a higher resolution of 240x180, higher dynamic range, lower power consumption and allows a concurrent readout of global shutter image frames, which are captured using the same photodiodes as for the DVS event generation. This frame output is not used for ELiSeD but only for ground truth measurements with the frame-based LSD algorithm).

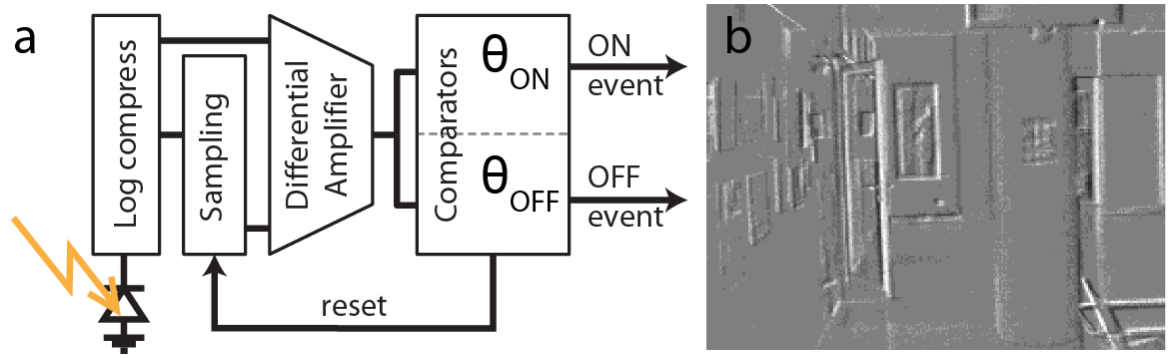

Fig. 1. a) Block diagram of the DVS pixel. b) Example output of the DVS events from the DAVIS240 sensor. The image is a 2D histogram of a 20ms time slice of accumulated ON events (in white) and OFF events (in black). A total of about 20k events are shown, color scale is 6 events for full black or full white.

A. *Related Work*

Due to a lack of event-based, low-level features which parametrize the event stream by assigning the events to spatial features, existing algorithms directly relate the events to trackers which can be classified by two groups: event clusters trackers and shape trackers.

Event cluster trackers [7]–[9] assign incoming events to the closest event cluster in the spatio-temporal neighbourhood. The position, orientation and size of these event clusters are temporally weighted averages of the assigned events. These algorithms are very cheap to compute and work reliably as long as the camera is fixed and tracked objects are continuously moving and spatially confined. Several robots with reaction latencies of under 5ms have been built using these trackers [9], [10] and typically the desktop/laptop CPU load in these applications is under 5%; for simple tracking even an embedded fixed-point microcontroller can suffice [10]. A more recent approach to cluster trackers approximates moving objects as spatially bivariate Gaussian distributions [11] or spatially connected sets of such distributions [12]. But since all these algorithms track event activity blobs and not specific spatial features, they are unselective and tend to merge trackers of arbitrary objects. For these reasons they are not appropriate for moving backgrounds or cluttered scenes.

Shape trackers use a pre-defined parameterization of the objects they track. These parametrizations range from lines [10], to arbitrary Gaussian kernels [11] or arbitrary pre-defined shapes [11], [13]. To track these shapes, the events are used to infer the most probable transformation of the parameters that describe the shape using various methods, such as an adapted Hough transform [10], a spatial probability measure [11] or an iterative closest point approach [13]. While shape trackers can robustly track even complicated shapes, they require prior knowledge on the scene content and the possibility to parametrize the objects or features of interest.

To improve the distinctiveness of cluster trackers or the generality of shape trackers, they should track specific spatial features. To achieve this, a solution to the event correspondence problem is needed as described in the following.

B. *Event Correspondence Problem*

In frame-based machine vision applications, such as visual odometry or structure from motion, spatial features must be matched across frames; which is known as motion correspondence problem. The high temporal resolution and continuous nature of the DVS events allow continuously tracking the position of a feature and thereby circumventing the motion correspondence problem. But another matching problem must be solved: To allow a continuous event-based position update of a feature, each event has to be attributed to a source of temporal contrast. Under the assumption of a constant scene illumination, a noise-free sensor, and spatial structures that do not change reflectance or shape, the only source of temporal contrast is relative motion of structures with spatial contrast according to the brightness constancy assumption and the resulting optical flow equation:

$$\frac{\partial I}{\partial x} \cdot v_x + \frac{\partial I}{\partial y} \cdot v_y + \frac{\partial I}{\partial t} = 0 \qquad (1)$$

where I is the (log) light intensity, v_x, v_y the two components of the velocity/optical flow and $\partial I/\partial t$ is the temporal contrast. Each DVS event encodes a step in log intensity $\Delta\log(I)$ and if an event could be attributed to a spatial structure this would allow inferring the motion of said structure. The event correspondence problem is therefore the following: which events in a stream of events correspond to the motion of the same spatial structure? Without direct access to images of the spatial structures or their spatial derivatives, e.g. through the frame readout in the DAVIS, these structures have to be inferred from the event stream.

The assumptions behind the proposed algorithm are that most events are generated by the relative motion of contours and that these contours can be modelled and parametrized as a set of piecewise linear segments. The proposed solution to the event correspondence problem is based on attributing the events to line segments. This results in a line segment tracking algorithm that has the generality of event cluster tracker as well as the specificity of a shape tracker. In

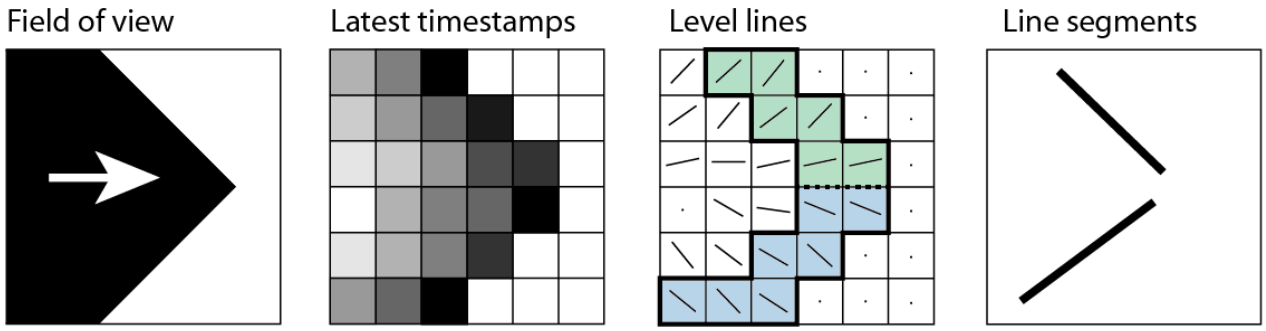

Fig. 2. ELiSeD algorithm overview: A corner with spatial contrast moves through the field of view and generates events with increasing timestamps (encoded from bright to dark pixels). By computing the level line orientation of this timestamp gradient, the pixels in the buffer (thick black border lines) can be clustered into support regions and line segments can be fitted. The green and blue regions show how clustered into support regions with similar timestamp gradient directions.

addition, this parametrization of the event stream into a set of line segments could eventually allow for feature descriptors and cross-stream matching through descriptions of local subsets of line segments.

C. Event Notation

Multiple publications on event-based algorithms use an event notation that defines an event as a function of its coordinates x and y, e.g. in the form of *p(x,y)* as well as its timestamp $Ev(p,t) = -1\ or\ 1$ depending on its polarity e.g. [13] which leaves the function undefined for most inputs. The notation used in this paper is more general and closer to the data representation used to communicate and process the events. An event *Ev* is understood as a tuple of an address *k* and timestamp *ts* with index *i*: $Ev_i = (k, ts)$. The address *k* carries all information about the sender; in the case of a DVS it contains *x*, *y* and *pol* (0 for ON and 1 for OFF) which are bitwise concatenated into a single address *k*. Functions *Addr*(*Ev*), *Ts*(*Ev*), *X*(*Ev*), *Y*(*Ev*), *Pol*(*Ev*) return the variables of the tuple and the function *Sign*(*Ev*) returns -1 for OFF and +1 for ON events.

III. Method

Line detection in images has been extensively researched in frame based computer vision and many algorithms have been developed around the Hough transform [14]. While the Hough transform allows simple line detection and has already been applied to events [10], it has several drawbacks when it comes to event-based line segments detection. To preserve the low latency of the events as well as the compactness of the data, any event-based algorithm is preferentially updated on each event. But Hough projection, peak detection and endpoint determination for each event are computationally expensive and require finicky selection of binning and peak selection criteria. For this reason, a more bottom-up approach for line segment detection has been used as starting point: the LSD line segment detector [15]. The basic idea behind the LSD is to compute the orientation of the spatial derivative for each pixel (called level lines) and cluster pixels with similar orientation into support regions which are used to fit a line segment. The following sections explain how the idea behind the LSD algorithm was adapted to DVS events to form the event-based line segment detector algorithm (**ELiSeD**). This algorithm is released as open-source code as described in the paper conclusion.

A. Event Clustering Criterion

ELiSeD should cluster only events from the same line segment. This clustering is achieved by attributing an orientation to each event and only clustering events of a similar orientation. This orientation is computed similar to [7] or [16] as edge detection on a map T_L that stores the latest event timestamps per pixel (Fig. 2: latest timestamps). The underlying principle is the following: if a sharp edge of sufficient spatial contrast moves through the field of view of the DVS, it will trigger one or multiple events per pixel and T_L then contains the information when the last edge passed a specific pixel. The entries in T_L along the contours of a moving object are similar and fall off smoothly in a direction perpendicular to the contour on one side, while falling off abruptly on the other side where the edge has not yet passed (Fig. 2, latest timestamps). The orientation of an edge can therefore be computed using the spatial derivative on T_L. In ELiSeD, for a stable and cheap edge detection, Sobel filters [17] SF^x, SF^y are employed to compute the orientation angle $\omega(Ev)$ for each event:

$$\omega(Ev) = \operatorname{atan2}(SF^y(T_L(x,y)), SF^x(T_L(x,y))) \qquad (2)$$

The Sobel operators SF^x and SF^y are 3x3 matrices that compute the *x* and *y* discretized gradients, smoothed in the perpendicular direction. The *atan2* computes the gradient direction from these timestamp surface gradients.

For a better performance ON and OFF events have separate T_L arrays and the T_L array used to compute the level line angle is chosen according to the event polarity. To

ALGORITHM 1: ELISED ALGORITHM

```
1:  while hasNextEvent(packet)
2:      Ev = getNextEvent(packet)
3:      removeEventFromPixel(buffer.getOldest())
4:      buffer.add(Ev)                                   Circular Buffer
5:      T_L(X(Ev), Y(Ev)) = Ts(Ev)
6:      ω(Ev)=computeAngle(T_L, X(Ev), Y(Ev))            According to (2)
7:      setAngle(LevelLine(X(Ev), Y(Ev)), ω)
8:      for neighbour:getNeighbourPixels(X(Ev), Y(Ev))
9:          if neighbour.hasBufferedEvents() &&
            angleDifference(neighbour.getAngle(), ω) < ρ
10:             candidates.add(neighbour)
11:         end if
12:     end for
13:     candidates.add(LevelLine(X(Ev), Y(Ev)))
14:     oldestSupport = findOldestSupport(candidates)
15:     if oldestSupport != null
16:         assignCandidates(oldestSupport, candidates)
17:         updateLineSupport(oldestSupport)
18:     else if size(candidates) > minNeighbours
19:         addNewLineSupport(candidates)
20:     end if
21:   candidates.clear()
22:   end while
```

improve the accuracy of ω, timestamps that are older than a given threshold (typically in range of 30ms to 100ms) are ignored. This threshold sets a lower bound on edge speeds that are detected. It could be made adaptive, e.g. by making the threshold inversely proportional to average pixel event rate.

B. Support Regions and Line Fitting

Similar to the original LSD algorithm, pixels with the same level line angle are clustered into line support regions. For each incoming event, the corresponding support regions are updated: After the level line angle ω is computed, the 8 neighbouring pixels are searched for other pixels with the same angle or an existing line support with the same orientation. If the averaged level line angle, i.e. the orientation of a support region is close enough (within a tolerance angle ρ) to the level line angle at the event position, the pixel is added to the support region. If a number of pixels (at least *minNeighbors*) have a similar level line angle ($\pm$ ρ, typically 23 deg) and none of them is assigned to a support region, the pixels form a new support region.

To make sure that support regions do not rely on obsolete data and grow infinitely large, the level line angle of pixels as well as their allocation to a support region must be purged. Purging is achieved by using a circular buffer (with typical length 2500 to 8000 events): Every time a new event is added, the oldest event is removed from the buffer, the level line angle is set to null and it is removed from any support region. By using a removal method that does not depend on time, the filter output becomes velocity independent: no matter how fast or slow a stimulus is moving, the tracking output is the same (as shown in Fig. 4) because it does not rely on a temporal window of constant length or decay constant for "forgetting" event-based data unlike [9]–[12]. The trade-off for using constant number of events in a single global buffer is that can lead to the buffer containing only a single or few fast moving edges, as discussed in the paper conclusion.

The original LSD algorithm fits a rectangular box covering the full line support region. This would require a bounding box update on every event added to or removed from a support region which is computationally expensive. The line segments in ELiSeD are therefore approximated by the major elliptic axis computed from the image moment of the support region [18] which can be computed efficiently according to [19].

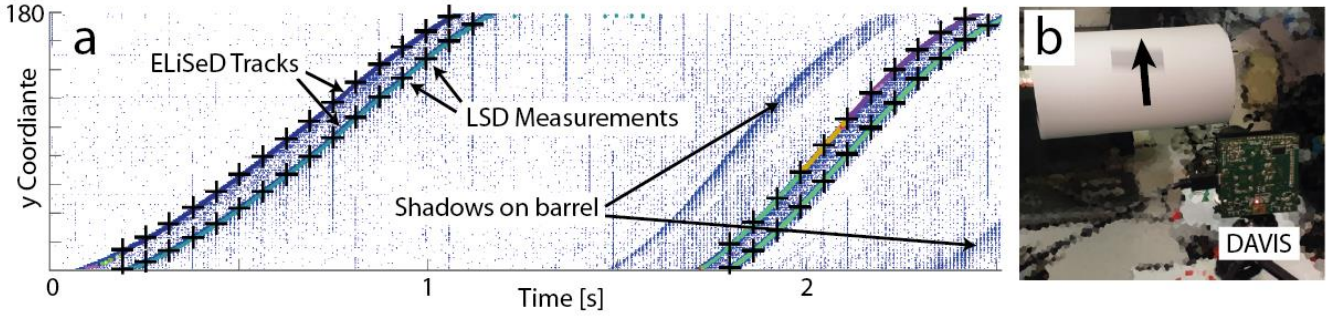

Fig. 3. ELiSeD tracking vs LSD tracking of a thick black line on a rotating barrel. a) DVS Events: blue, ELiSeD Segment traces: coloured according to ID, LSD performed on DAVIS frames: black crosses. b) Setup with rolling barrel spinning upwards along arrow.

C. The ELiSeD Algorithm

The ELiSeD algorithm is described in pseudocode (ALGORITHM 1). The main elements in the algorithm are an event packet *packet* containing the latest events, a circular buffer *buffer* that stores the coordinates of the latest *n* events, a 2D array T_L containing the latest timestamp per pixel, a set of the active line supports S_A containing all pixels assigned to a given line support and a 2D array A_S containing following information for each pixel: the level line angle ω, the number of buffered events with the pixel coordinate and the assigned support region. For each event the 8 neighbouring pixels (*neighbour*) are searched for *candidate* pixels with the right orientation which are then allocated to the oldest line support regions among them.

The events are processed in packets and for each new event, the oldest event in the circular buffer is removed from TL ("removeEventFromPixel") and if there are no more events with the according pixel coordinate in the buffer, the orientation of this pixel is set to null and it is removed from all support regions. In the next steps the new event is buffered and it is used to compute the level line angle ω of the pixel with the event coordinates. Then the algorithm iterates over the 8 pixel neighbourhood of the pixel to collect candidates with a level line angle that does not differ more than ρ (typically 23 deg) from the computed ω. If the neighbour is already assigned to a support region, "getAngle()" returns its orientation. From these candidates the oldest support region is determined and all candidates including their support regions are assigned.

To prevent support regions growing too large through accidental merging with other support regions, after every packet the width of all support regions is assessed and any that are above a certain threshold (typically 15 pixels) are split.

IV. Experiments

To assess the performance of the proposed algorithm, several experiments have been conducted with the events of a DAVIS240 camera.

A. Accuracy

To assess the accuracy of the line feature, two horizontal black bars of different length were placed on a white barrel rotating upwards along the longitudinal axis (Fig. 3). The ELiSeD output was compared to the output of the LSD algorithm by capturing global shutter frames for the LSD algorithm in parallel to recording the events for the ELiSeD

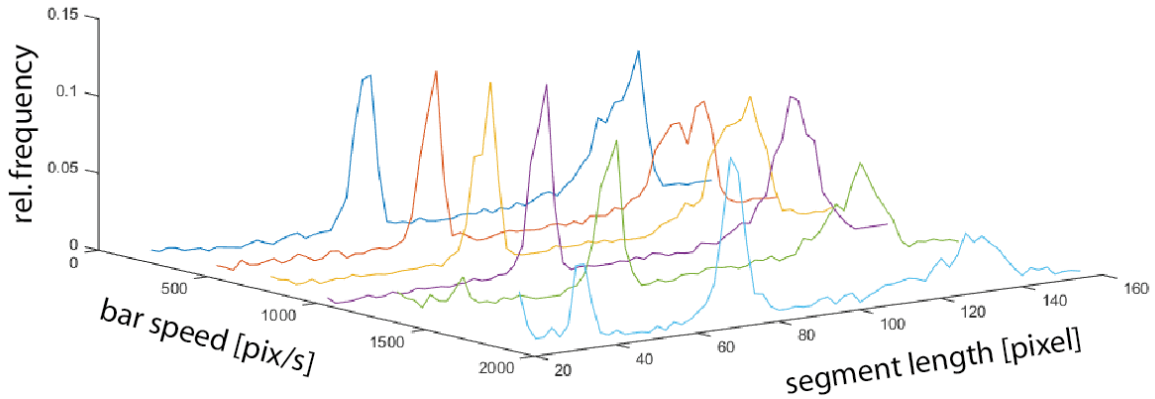

Fig. 4. Frequency of ELiSeD segment lengths for different bar speeds.

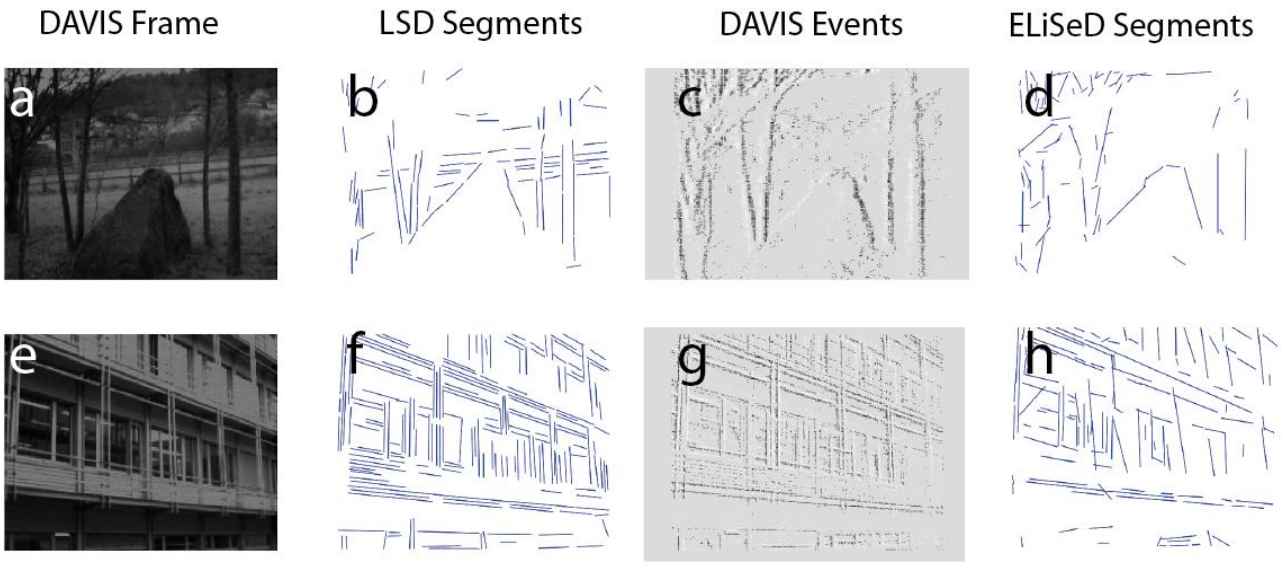

Fig. 5. DAVIS frames / events and LSD / ELiSeD line segments ot two scenes: a)-d) outdoor scene with stone and trees, e)-h) building frontage with balconies

algorithm. The accuracy was measured by computing the difference in *y* coordinates of the horizontal LSD segments and the closest ELiSeD segment, resulting in an accuracy of 1.36 pixels. This offset results from the buffer-induced lag as well as synchronization issues between event stream and frames.

B. Speed

The ELiSeD algorithm ran on a QuadCore Intel Core i7 Q820 1.73GHz laptop in real-time for a single black bar up to rotation frequency of 2Hz which corresponds to about 1m/s surface speed at 50cm distance (f=12mm) or 1300pix/s. Fig. 4 shows how the accuracy of the tracker (same setup as in Fig. 3) degrades when the barrel spins too fast. Ideally each histogram should have two peaks at about 70 and 130 pixel long segments. When the sensor produces too many events (500,000 events per second), they cannot be processed anymore and the longer of the two black bars (~130 pixel wide) is only rarely covered with a full line segment in the cyan histogram for 2000 pixels/second bar speed.

C. Line Segment Coverage

To compare the performance of the ELiSeD algorithm with the original LSD algorithm, Fig. 5 shows how the static image frames of the DAVIS from two different scenes (a,e) were used to detect line segments using the conventional LSD (b,f), at the same time as ELiSeD was used to detect segments (d,h) using the events (c,g). ELiSeD does not detect and track small line features but it can still detect the most prominent contours in natural scenes as well as in man-made environments with more straight lines. It can further be noted that only line segments perpendicular to the camera motion can be detected because spatial contrast close to parallel to the motion generates fewer events.

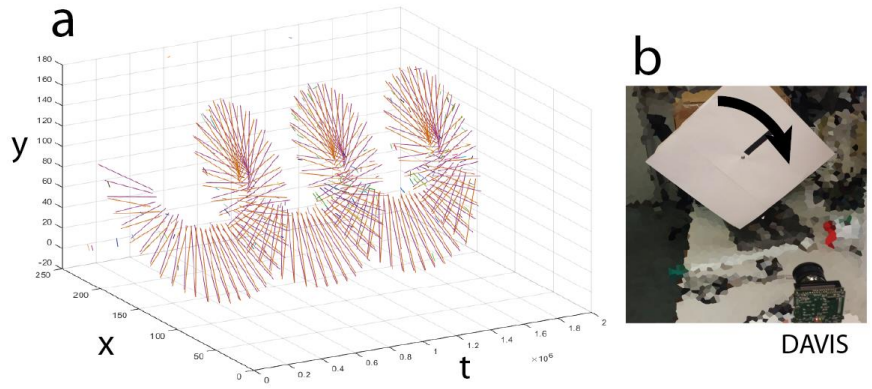

Fig. 6. a) Lifetime of ELiSeD line segments of a rotating disk b) Setup.

D. Stability

For many applications, it is important that line segments can be tracked stably over sufficient time to exploit the motion correspondences. Tracker stability is demonstrated using a spinning disk with a black bar as shown in Fig. 6. The position of the line segment was logged every 5000 events and the line segments are coloured according to their ID. It can be seen that the color of the segment tracking the main bar (red) does not change over time and the contour of the bar can be continuously tracked. The tracker was capable of tracking the rotating stimulus at speeds up to about 720 deg/s in real-time on said laptop, limited by the (currently rather inefficient) Java implementation.

CONCLUSTION AND OUTLOOK

The proposed algorithm for an event-based line segment feature detector and tracker solves the event correspondence problem by assigning DVS events to a set of spatial line segment features. This Event-based Line Segment Detector (ELiSeD) algorithm parametrizes the contours of objects and shapes as a set of line segments extracted from the events of a DVS. ELiSeD allows circumventing the motion correspondence problem by continuously updating the position of spatial features. The line features are distinct enough to avoid unwanted merging so that they can be used as trackers in scenes with moving background and without the requirement of predefining the shape to be tracked.

The implementation is open source code in the jAER project [20] as the Java package ch.unizh.ini.jaer.projects.elised. This package also has suggestions for parameter settings.

A first step to improve the stability of the algorithm will be to replace the static global circular buffer with a more advanced, time-independent way of buffering the relevant events. A dynamic event buffering method that scales the buffer size will allow having the same performance independent of the amount of spatial contrast in a scene. A local buffering method will allow handling objects moving at different velocities (otherwise fast objects dominate the buffer contents), e.g. [21]. To reduce the computational load simpler methods to perform the event-based level line extraction could be used such as in [22].

Apart from improving the algorithm, the line segments features could be applied to Simultaneous Localization and Mapping (SLAM) (e.g. as in [23] or [24]) for a deeper evaluation of its generality and robustness.

ACKNOWLEDGEMENTS

The authors would like to thank anonymous reviewers for their useful suggestions for improved presentation.

REFERENCES

[1] A. Censi, “Efficient Neuromorphic Optomotor Heading Regulation,” in *The 2015 American Control Conference*, Chicago, USA, 2015.

[2] J. Carmack, “Latency Mitigation Strategies,” *Twenty Milliseconds*, 25-Oct-2014.

[3] P. Lichtsteiner, C. Posch, and T. Delbrück, “A 128 x 128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor,” *IEEE J Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[4] T. Delbruck, B. Linares-Barranco, E. Culurciello, and C. Posch, “Activity-Driven, Event-Based Vision Sensors,” in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, Paris, 2010, pp. 2426–2429.

[5] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, “A 240x180 130 dB 3 us Latency Global Shutter Spatiotemporal Vision Sensor,” *IEEE J. Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, Oct. 2014.

[6] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas, Eds., *Event-Based Neuromorphic Systems*. John Wiley and Sons Ltd., UK, 2015.

[7] T. Delbruck, “Frame-free dynamic digital vision,” in *Proceedings of Intl. Symp. on Secure-Life Electronics*, Tokyo, Japan, 2008, vol. 1, pp. 21–26.

[8] S. Schraml, A. N. Belbachir, N. Milosevic, and P. Schön, “Live demonstration: Dynamic stereo vision system for real-time tracking,” in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 1408–1408.

[9] T. Delbruck and M. Lang, “Robotic Goalie with 3ms Reaction Time at 4% CPU Load Using Event-Based Dynamic Vision Sensor,” *Front. Neurosci.*, vol. 7, p. 223, Nov. 2013.

[10] J. Conradt, M. Cook, R. Berner, P. Lichtsteiner, R. J. Douglas, and T. Delbruck, “A Pencil Balancing Robot Using a Pair of AER Dynamic Vision Sensors,” in *IEEE International Symposium on Circuits and Systems (ISCAS) 2009*, Taipei, 2009, pp. 781–784.

[11] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat, and R. Benosman, “Asynchronous Event-Based Multikernel Algorithm for High-Speed Visual Features Tracking,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. PP, no. 99, pp. 1–1, 2014.

[12] D. R. Valeiras, X. Lagorce, X. Clady, C. Bartolozzi, S.-H. Ieng, and R. Benosman, “An Asynchronous Neuromorphic Event-Driven Visual Part-Based Shape Tracking,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. PP, no. 99, pp. 1–1, 2015.

[13] Z. Ni, A. Bolopion, J. Agnus, R. Benosman, and S. Regnier, “Asynchronous Event-Based Visual Shape Tracking for Stable Haptic Feedback in Microrobotics,” *IEEE Trans. Robot.*, vol. 28, no. 5, pp. 1081–1089, Oct. 2012.

[14] V. F. Leavers, “Which Hough Transform?,” *CVGIP Image Underst.*, vol. 58, no. 2, pp. 250–264, Sep. 1993.

[15] R. Grompone von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall, “LSD: a Line Segment Detector,” *Image Process. Line*, vol. 2, pp. 35–55, Mar. 2012.

[16] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, “Event-Based Visual Flow,” *IEEE Trans. Neural Netw. Learn. Syst.*, 2013.

[17] I. Sobel and G. Feldman, “A 3x3 Isotropic Gradient Operator for Image Processing,” *Pattern Classif. Scene Anal.*, pp. 271–2, 1968.

[18] B. B. Chaudhuri and G. P. Samanta, “Elliptic fit of objects in two and three dimensions by moment of inertia optimization,” *Pattern Recognit. Lett.*, vol. 12, no. 1, pp. 1–7, Jan. 1991.

[19] L. Rocha, L. Velho, and P. C. P. Carvalho, “Image moments-based structuring and tracking of objects,” in *XV Brazilian Symposium on Computer Graphics and Image Processing, 2002. Proceedings*, 2002, pp. 99–105.

[20] “jAER Open Source Project,” *jAER Open Source Project*, 23-Mar-2007. [Online]. Available: http://jaerproject.org. [Accessed: 23-May-2016].

[21] E. Mueggler, C. Forster, N. Baumli, and G. Gallego, “Lifetime Estimation of Events from Dynamic Vision Sensors,” in *International Conference on Robotics and Automation ICRA*, Seattle, WA, USA, 2015.

[22] K. Lee, H. Ryu, S. Park, J. H. Lee, P. Park, C.-W. Shin, J. Woo, T.-C. Kim, and B.-C. Kang, “Four DoF gesture recognition with an event-based image sensor,” in *2012 IEEE 1st Global Conference on Consumer Electronics (GCCE)*, 2012, pp. 293–294.

[23] P. Smith, I. Reid, and A. Davison, “Real-Time Monocular SLAM with Straight Lines,” presented at the British Machine Vision Conference, 2006, vol. 1, pp. 17–26.

[24] L. Zhang and R. Koch, “Hand-Held Monocular SLAM Based on Line Segments,” in *Machine Vision and Image Processing Conference (IMVIP), 2011 Irish*, 2011, pp. 7–14.