Event-Based Line Fitting and Segment Detection using a Neuromorphic Visual Sensor

David Reverter Valeiras, Xavier Clady, Sio-Hoi Ieng and Ryad Benosman

***Abstract*—This paper introduces an event-based luminance-free algorithm for line and segment detection from the output of asynchronous event-based neuromorphic retinas. These recent biomimetic vision sensors are composed of autonomous pixels, each of them asynchronously generating visual events that encode relative changes in pixels' illumination at high temporal resolutions. The proposed algorithm is based on an iterative event-based weighted least squares fitting, and it is consequently well suited to the high temporal resolution and asynchronous acquisition of neuromorphic cameras: parameters of a current line are updated for each event attributed (i.e. spatio-temporally close) to it, while implicitly forgetting the contribution of older events according to a speed-tuned exponentially decaying function. A detection is validated if a measure of activity, i.e. implicit measure of the number of contributing events and using the same decay function, exceeds a given threshold. The speed-tuned decreasing function is based on a measure of the apparent motion, i.e. the optical flow computed around each event. This latter ensures that the algorithm behaves independently of the edges' dynamics. Line segments are then extracted from the lines, allowing for the tracking of the corresponding endpoints. We provide experiments showing the accuracy of our algorithm and study the influence of the apparent velocity and relative orientation of the observed edges. Finally, evaluations of its computational efficiency show that this algorithm can be envisioned for high-speed applications, such as vision-based robotic navigation.**

***Index Terms*—Line Detection, Segment Detection, Neuromorphic Sensing, Event-based Vision.**

I. Introduction

This paper introduces a new methodology to extract and estimate parameters of lines and segments present in a visual scene acquired using a neuromorphic event-based camera. Neuromorphic cameras are a novel type of asynchronous biomimetic vision sensors, often referred to as "silicon retinas" [1]. Neuromorphic cameras have convincingly demonstrated their potential for energy efficiency and high temporal resolution (from 1 µs - 1 ms) allowing to rethink most computer vision algorithms to operate in the event-based framework in real-time. Since the pioneering work of Mahowald [2] a variety of these event-based devices have been designed, including gradient-based sensors sensitive to static edges [3], temporal contrast vision sensors sensitive to relative luminance change [4]-[6], edge-orientation sensitive devices and optical-flow sensors [7]-[8]. A complete review of the history and existing sensors can be found in [9]. For a recent review of neuromorphic systems, sensors and processing, the interested reader can refer to [10]. The Asynchronous Time-based Image Sensor (ATIS) used in this work is a new type of clockless neuromorphic camera outputting events rather than frames as shown in Fig. 1. Each pixel contains a change detector circuit that asynchronously and autonomously generates events. Each event signifies a change in log intensity of the luminance at a spatial location on the focal plane at 1 µs temporal precision. A polarity is associated to each event depending on whether the luminance increases or decreases. The ATIS camera also provides an asynchronous absolute measurement of light intensity encoding in the time domain. But this functionality is not being used in the presented work; interested readers can refer to [6]. Unlike conventional cameras, neuromorphic vision sensors do not sample data at an arbitrary frequency imposed by some artificial clock signal that has no relation to the source of the visual information. This frame-free approach reduces the amount of redundant information generated by conventional sensors, allowing for a drastic increase of its temporal resolution while allowing for real time processing at several hundreds kHz at a low energy cost.

Extracting lines (and segments) from man-made environments provides indispensable low level features for vision-based systems. They are frequently used in robotics as features for visual odometry [11] and vision-based structure from motion (SFM) [12]-[13]. More recently, the two problems have been combined to be used in Simultaneous Localization and Mapping (SLAM) [14]-[16]. Various approaches for the detection of lines and segments have been proposed starting for the textbook classic Hough transform [17] that can be generalized to the detection of generic shapes [18]. This technique is still widely used and studied as shown in the recent State-of-Art proposed in [19]. Other classical approaches include gradient-based methods [20] or statistical-based techniques [21]. In the field of segment detection, LSD [22] is currently among the most used algorithms.

Recently, an event-based segment detector has been published in [23], called Event-Based Line Segment Detector (ELiSeD) and derived from the LSD [22]. The reported accuracy, when tracking a single line, is about 1.36 pixels compared with the original LSD algorithm. The ELiSeD method requires events to be stored on a circular buffer of a fixed size (they report typical sizes of 2500 to 8000 events). Considering a buffer is implicitly building a frame with the main disadvantage of losing the ability of the algorithm to adapt to velocities in the scene, as the number of events is related to the amount of motion. As a result, fast moving objects, that generate globally more events at constant contrast, tend to dominate the buffer contents.

To address these limitations, we propose an event-based line and segment detector using an iterative single event update to determine the parameters of a given line. The method uses an iterative weighted least squares fitting for each incoming event

attributed (i.e. spatio-temporally close) to a line. The weights of past events follow a speed-tuned exponentially decaying function, which makes the method velocity-independent. We will also extend the line detection technique to detect segments in order to provide pixel-wise feature detection. This is useful for many computer vision algorithms that require pixel-wise feature detection, identification and tracking (e.g. [24]).

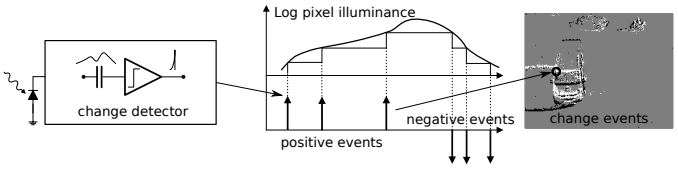

Fig. 1. Each pixel asynchronously generates events when a sufficient amount of change in the log of the luminance is detected at the corresponding position on the focal plane. These events are "positive" if the luminance increased and "negative" if it decreased.

II. Event-Based Line Detection

A stream of asynchronous events generated by a neuromorphic camera can be mathematically described as: $\mathbf{e}_k = [\boldsymbol{u}_k^T, t_k, p_k]^T$ a quadruplet describing an event occurring at time t_k at the position $\boldsymbol{u}_k = [x_k, y_k]^T$ in the focal plane. The two possible values for p_k are 1 or -1, depending on whether a positive or negative change of luminance has been detected, while the time t_k is usually referred to as the *timestamp* of the event. We are looking for the set of lines that best fits the stream of recent past events, in the sense of minimizing the sum of the squared distances between events spatial locations and lines.

A. Event-Based Visual-Flow

The first step to track lines from events is to compute the event based visual flow for each incoming event. We apply the technique described in [25], where the visual flow is obtained from the normal to a 3D plane locally approximating the spatio-temporal surface described by incoming events. In the standard plane parametrization of $Ax + By + t + C = 0$, the normal to a plane defined by data $[x, y, t]^T$ is directly related to the parameters A and B. To estimate these parameters, we apply a least squares fitting algorithm on the centered data to avoid ill conditioned systems [26]. The plane equation is then:

$$\widetilde{t} = A\widetilde{x} + B\widetilde{y}, \tag{1}$$

where the tilde means that the average values are subtracted from each variable. We consider the most recent events in a given spatial neighborhood of the current event and we denote $\mathcal{I}_k$ the set of indices identifying these events (i.e. if $i \in \mathcal{I}_k$ then $\mathbf{e}_i$ is one of the most recent events in a spatial neighborhood of $\mathbf{e}_k$).

This yields the linear system:

$$\begin{bmatrix} \sum_{i\in\mathcal{I}_k} \widetilde{x}_i^2 & \sum_{i\in\mathcal{I}_k} \widetilde{x}_i\widetilde{y}_i \\ \sum_{i\in\mathcal{I}_k} \widetilde{x}_i\widetilde{y}_i & \sum_{i\in\mathcal{I}_k} \widetilde{y}_i^2 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} \sum_{i\in\mathcal{I}_k} \widetilde{x}_i\widetilde{t}_i \\ \sum_{i\in\mathcal{I}_k} \widetilde{y}_i\widetilde{t}_i \end{bmatrix}. \tag{2}$$

Parameters A and B can be calculated by inverting the 2×2 matrix in the left hand side of the equation, this is however a very inefficient technique as inverting a matrix for each incoming event might prevent from operating in real time. Since this is a 2×2 linear system, when the system matrix is invertible we can apply Cramer's rule to obtain a closed form solution, allowing us to compute A and B. The velocity vector is, according to [25], given by:

$$\boldsymbol{v}_k = \frac{1}{A^2 + B^2} \begin{bmatrix} -A \\ -B \end{bmatrix} \tag{3}$$

From this result we define an "oriented" event $\mathbf{o}_k = [\boldsymbol{u}_k^T, \boldsymbol{v}_k^T, t_k, p_k]^T$, as an event augmented with the velocity $\boldsymbol{v}_k$.

B. Event-Based Least Squares Line Fitting

1) Line Parametrization: we derive an iterative event-based rewriting of the least squares fitting of lines with perpendicular offsets [27]. Lines in 2D space are defined by two parameters, we choose the ρ-θ (or polar) parametrization [28] that avoids infinite or close to infinite slopes that one can encounter when using the slope-intercept parametrization [29]. We define a given line model i as $\mathcal{L}^{(i)}\big(\rho_k^{(i)}, \theta_k^{(i)}\big)$, where $\rho_k^{(i)}$ is the distance from the line to the origin, and $\theta_k^{(i)}$ the angle between the normal $\boldsymbol{n}_k^{(i)}$ to the line and the horizontal (see Fig. 2). The subindex k relates the line to time, i.e. $\rho_k^{(i)}$ is the angle of line i at time t_k, etc. The equation of the line is then:

$$x\cos(\theta_k^{(i)}) + y\sin(\theta_k^{(i)}) - \rho_k^{(i)} = 0, \tag{4}$$

where $\theta_k^{(i)} \in [-\pi/2, \pi/2]$ and the unit vector normal to the line $\boldsymbol{n}_k^{(i)}$ is:

$$\boldsymbol{n}_k^{(i)} = \big[\cos(\theta_k^{(i)}),\ \sin(\theta_k^{(i)})\big]^T. \tag{5}$$

Remark: while lines are defined by the angle $\theta_k^{(i)}$, we will not need to calculate it explicitly. The algorithm can be directly applied on $\cos(\theta_k^{(i)})$, $\sin(\theta_k^{(i)})$ as it will be shown in section II-B6.

2) Activity: each possible line model has a certain level of activity $\mathcal{A}_k^{(i)}$ as in [30]. The activity of every model is updated with the incoming events $\mathbf{e}_k$, following a speed-tuned exponential decay function:

$$\mathcal{A}_k^{(i)} = \begin{cases} \mathcal{A}_{k-1}^{(i)} e^{-\|\boldsymbol{v}_k\|\Delta t_k} + 1 & \text{if } \mathbf{e}_k \text{ is assigned to } \mathcal{L}^{(i)}, \\ \mathcal{A}_{k-1}^{(i)} e^{-\|\boldsymbol{v}_k\|\Delta t_k} & \text{otherwiseM} \end{cases} \tag{6}$$

where $\Delta t_k = t_k - t_{k-1}$. Here, we choose a speed-tuned decreasing strategy, which makes the activity of the lines independent to their respective velocities. This is an important advantage over the fixed decreasing strategies, as it provides an automatic and adaptive way to characterize each line. A more detailed explanation of this strategy is discussed in Section II-C.

If the activity $\mathcal{A}_k^{(i)}$ of a line model $\mathcal{L}^{(i)}$ is greater than a predefined threshold, then $\mathcal{L}^{(i)}$ is said to be *visible*, and a line is assumed to be actually present at that position.

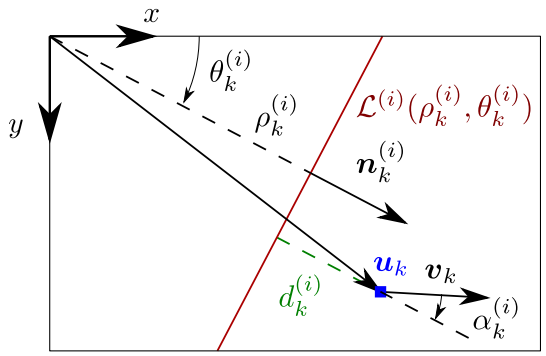

Fig. 2. A line, identified by its index i, is denoted $\mathcal{L}^{(i)}\left(\rho_k^{(i)}, \theta_k^{(i)}\right)$ and defined by two parameters: the distance $\rho_k^{(i)}$ between the line and the origin and the angle $\theta_k^{(i)}$ between the normal to the line $\boldsymbol{n}_k^{(i)}$ and the horizontal. Incoming events are assigned to previously existing lines based on two conditions: the euclidean distance $d_k^{(i)}$ between the line and the position of the event $\boldsymbol{u}_k$, and the angular distance $\alpha_k^{(i)}$ between the normal to the line $\boldsymbol{n}_k^{(i)}$ and the visual flow of the event $\boldsymbol{v}_k$.

3) Assignment of events: Let $\mathbf{o}_k = [\boldsymbol{u}_k^T, \boldsymbol{v}_k^T, t_k, p_k]^T$ be an oriented event occurring at time t_k at the position $\boldsymbol{u}_k$ on the focal plane, with normal flow $\boldsymbol{v}_k$. The assignment of $\mathbf{o}_k$ to a line is based on two criteria:

- The euclidean distance $d_k^{(i)}$ from the event to the line has to be smaller than a threshold d_{max}.
- The angle $\alpha_k^{(i)}$ between the normal to the line $\boldsymbol{n}_k^{(i)}$ and the visual flow of $\boldsymbol{v}_k$ must be smaller than a threshold α_{max} (see Fig. 2), in order to assure orthogonality of the flow to the line.

The two criteria are translated into the following inequalities that an event has to satisfy in order to be assigned to a line $\mathcal{L}^{(i)}$:

$$\begin{cases} d_k^{(i)} = \left|\boldsymbol{u}_k^T \boldsymbol{n}_{k-1}^{(i)} - \rho_{k-1}^{(i)}\right| < d_{max}, \\ \left|\cos(\alpha_k^{(i)})\right| = \left|\boldsymbol{v}_k^T \boldsymbol{n}_{k-1}^{(i)}\right| > \cos(\alpha_{max}). \end{cases} \tag{7}$$

If several line models verify these two conditions, the event is simply assigned to the line with the highest activity. If an oriented event does not satisfy any existing line, a new one is initialized.

4) Initialization of line models: the form of a line indexed by i created for an incoming oriented event $\mathbf{o}_k$, is given by:

$$\begin{cases} \boldsymbol{n}_k^{(i)} = \boldsymbol{v}_k, \\ \rho_k^{(i)} = \boldsymbol{u}_k^T \boldsymbol{v}_k. \end{cases} \tag{8}$$

To limit the computational time required by the algorithm we fix a maximum of N line models to be tracked simultaneously. If N line models have already been created, then the newly created one replaces the line with the lowest activity.

5) Optimal parameters: when an event $\mathbf{o}_k$ is assigned to a line, the parameters of the line are updated accordingly. The new optimal parameters must minimize the sum of the squared distances between the line and the past events assigned to it. As in [24], these distances are weighted in order to give greater importance to the most recent events. Hence, the function $E_k^{(i)}(\theta, \rho)$ to be minimized is:

$$E_k^{(i)}(\theta, \rho) = \frac{1}{\Omega_k} \sum_{j=0}^{k} \omega_{k,j} \big(x_j \cos(\theta) + y_j \sin(\theta) - \rho\big)^2, \tag{9}$$

where $\omega_{k,j}$ is the weight of the j^{th} event assigned to the i^{th} line, at time t_k. The weights verify:

$$\omega_{k,j} \geq 0, \quad \forall j \in \{0, 1, ..., k\}, \tag{10}$$

with:

$$\Omega_k = \sum_{j=0}^{k} \omega_{k,j}. \tag{11}$$

We can define several strategies to set the values of $\omega_{k,j}$, we will always chose the one providing greater importance to the most recent events. The weighting strategy used in this work is given in (16).

To minimize $E_k^{(i)}$ w.r.t. ρ and θ, we look for the two values that cancel the respective partial derivatives of $E_k^{(i)}$. The optimal $\rho_k^{(i)}$ is then obtained as:

$$\rho_k^{(i)} = \frac{\widehat{x}_k^{(i)} \cos(\theta) + \widehat{y}_k^{(i)} \sin(\theta)}{\Omega_k}, \tag{12}$$

while $\theta_k^{(i)}$ is deduced after simplifying the vanishing derivative into the equation:

$$a_k \sin\left(2\theta_k^{(i)}\right) + b_k \cos\left(2\theta_k^{(i)}\right) = 0, \tag{13}$$

where:

$$\begin{aligned} a_k &= \Omega_k \big(\widehat{xy}_k^{(i)} - \widehat{x}\widehat{x}_k^{(i)}\big) + \big(\widehat{x}_k^{(i)}\big)^2 - \big(\widehat{y}_k^{(i)}\big)^2, \\ b_k &= 2\big(\Omega_k \widehat{xy}_k^{(i)} - \widehat{x}_k^{(i)} \widehat{y}_k^{(i)}\big). \end{aligned} \tag{14}$$

Here, $\widehat{x}_k^{(i)}$, $\widehat{y}_k^{(i)}$, $\widehat{xy}_k^{(i)}$, ... denote the weighted sum of the corresponding coordinates of the events previously assigned to the line:

$$\begin{cases} \widehat{x}_k^{(i)} = \sum_{j=0}^{k} w_{k,j} x_j, & \widehat{y}_k^{(i)} = \sum_{j=0}^{k} w_{k,j} y_j \\ \widehat{x}\widehat{x}_k^{(i)} = \sum_{j=0}^{k} w_{k,j} x_j^2, & \widehat{y}\widehat{y}_k^{(i)} = \sum_{j=0}^{k} w_{k,j} y_j^2 \\ \widehat{xy}_k^{(i)} = \sum_{j=0}^{k} w_{k,j} x_j y_j. \end{cases} \tag{15}$$

We will refer to these values as the *auxiliary parameters* of a line, which are required to compute its optimal ρ, θ.

Let us note that the development so far is independent of the weighting strategy being used. In the following, we will choose the weights to follow a speed-tuned exponentially decaying strategy as in [31]:

$$w_{k,j} = \begin{cases} \prod_{i=j+1}^{k} e^{-\|\boldsymbol{v}_i\| \Delta t_i} = e^{-\|\boldsymbol{v}_k\| \Delta t_k} \omega_{k-1,j}, & \text{if } j < k \\ 1, & \text{if } j = k. \end{cases} \tag{16}$$

According to this strategy, similar weights (close to 1, because $(t_k - t_j) \simeq 0$) will be associated to events $\mathbf{e}_j$ (quasi)-simultaneously occurring at the last event's timing (t_k) and belonging to the current line, while the weights of older events corresponding to older locations of the line will rapidly tend to 0. The speed-tuning automatically adapts the decay according to the speed of the contour line (see Section II-C).

As in the Equation (6) about activity, this weighting strategy allows us to compute the auxiliary parameters in the following iterative form, where $\delta_k = e^{-\|\boldsymbol{v}_k\|\Delta t_k}$:

$$\begin{cases} \widehat{x}_k^{(i)} \approx \delta_k \widehat{x}_{k-1}^{(i)} + x_k \\ \widehat{y}_k^{(i)} \approx \delta_k \widehat{y}_{k-1}^{(i)} + y_k \\ \widehat{xx}_k^{(i)} \approx \delta_k \widehat{xx}_{k-1}^{(i)} + x_k^2 \\ \widehat{yy}_k^{(i)} \approx \delta_k \widehat{yy}_{k-1}^{(i)} + y_k^2 \\ \widehat{xy}_k^{(i)} \approx \delta_k \widehat{xy}_{k-1}^{(i)} + x_k y_k \end{cases} \tag{17}$$

Additionally, when applying this set of weights we obtain:

$$\Omega_k = \mathcal{A}_k^{(i)}. \tag{18}$$

Let us note that expressions in (17) still hold when the past events have been assigned to different lines, since the auxiliary parameters of a given model are only updated when an event is assigned to it.

6) Optimization strategy: as previously stated, the actual value of $\theta_k^{(i)}$ is never directly required. Instead, we just need its sinus and cosinus. This avoids the computation of an arctangent, a sinus and a cosinus, at the cost of computing three square roots. We are updating the parameters of lines with every event assigned to them, and consequently it is of great importance to limit the number of operations carried out for every incoming event. As shown in the Appendix A, from (13) we obtain:

$$\sin\theta_k^{(i)} = \pm\sqrt{\frac{1-\beta_k}{2}}, \quad \cos\theta_k^{(i)} = +\sqrt{\frac{1+\beta_k}{2}}, \tag{19}$$

where:

$$\beta_k = \pm\sqrt{\frac{a_k^2}{a_k^2+b_k^2}}. \tag{20}$$

Here, we are just keeping the positive solution for $\cos\theta_k^{(i)}$, because $\theta \in [-\pi/2,\ \pi/2]$. This yields a total of four possible combinations for $\sin\theta_k^{(i)}$, $\cos\theta_k^{(i)}$. To disambiguate the right solution from the four possible ones, a procedure is given in the Appendix B. Let us also remark that these equations are always well defined, since $\frac{a_k^2}{a_k^2+b_k^2} \geq 0$ and $\beta_k \leq 1$ for all values of a_k, b_k.

C. About speed-tuned strategy

We derive a speed-tuned decaying weighting function to provide a life time of a given contour inspired from [31]. The lifetime of a contour is related to the normal velocity, i.e. the inverse of the time during which the contour travels through a pixel. However, keeping exactly the same weighting strategy would require to store all the events assigned to the line (or at least the ones with significant weights). This would be non-efficient in terms of memory and computation time consumption.

Using Equation 16 we can derive an iterative algorithm. The decreasing weighting strategy is here applied to the previous model line (computed at time t_{k-1}) according to its synchrony measurement ($e^{-\|\boldsymbol{v}_k\|\Delta t_k}$, see Eq. 16) with the current event (occurring at time t_k), and not to the events ($\mathbf{e}_j$, contributing to the current model line) according to their synchrony measurements (as in [31]): if the model line is "outdated" in comparison with the current event, this latter weighs relatively more in the updating computation of the new line model.

D. Event-Based Line Detection Algorithm

A summary of the complete event-based line detection procedure is given in Algorithm 1.

Algorithm 1 Event-Based Line Detection

Require: $\mathbf{o}_k = [\boldsymbol{u}_k^T, \boldsymbol{v}_k^T, t_k, p_k]^T, \ \forall k \geq 0$

Ensure: $\rho_k^{(i)}, \theta_k^{(i)} \ \ \forall i = 1...N$

```
for every oriented event o_k do
  for every line L^(i) do
    Apply the decay to A_k^(i) using (6)
  end for
  if ∃ any lines verifying (7) then
    i ← arg max_j A_k^(j) s.t. L^(j) verifies (7)
    A_k^(i) ← A_k^(i) + 1
    Update sin θ_k^(i), cos θ_k^(i) and ρ_k^(i) using (19) and (12)
    if A_k^(i) > A_up^(L) then
      Output the line
    end if
  else
    Initialize a new line model using (8)
  end if
end for
```

III. Event-Based Segment Detection

Segments, from a line, are detected by localizing discontinuities which are actually the segments' endpoints. Hence, we add to the line detection algorithm some procedure to output the positions of these endpoints at the same time the line is detected and tracked. In our implementation, the algorithm produces a pair of *endpoint events* for each segment. These events signal the position in space and time of the endpoints. As such, the output of this segment detector is a stream of endpoint events that can be used as feature for matching and learning tasks.

A. Activity of each pixel

Let us define, for each line $\mathcal{L}^{(i)}$, two vectors $\boldsymbol{\mathcal{X}}_k^{(i)} \in \mathbb{R}^w$ and $\boldsymbol{\mathcal{Y}}_k^{(i)} \in \mathbb{R}^h$. These vectors contain the activity of each pixel of the line, projected on the x and y axis respectively, where w, h are the width and height of the sensor. The activity of each pixel is increased whenever an event is assigned to the line at

a distance smaller than l pixels, and it follows a speed-tuned exponential decay afterwards. Thus, when an oriented event $\mathbf{o}_k = [\boldsymbol{u}_k^T, \boldsymbol{v}_k^T, t_k, p_k]^T$ is assigned to a line $\mathcal{L}^{(i)}$, we update $\boldsymbol{\mathcal{X}}_k^{(i)}$ and $\boldsymbol{\mathcal{Y}}_k^{(i)}$ according to:

$$\boldsymbol{\mathcal{X}}_k^{(i)}(x) = \begin{cases} \boldsymbol{\mathcal{X}}_{k-1}^{(i)}(x) e^{-\|\boldsymbol{v}_k\|\Delta t_k} + 1 & \text{if } |x - x_k| < l, \\ \boldsymbol{\mathcal{X}}_{k-1}^{(i)}(x) e^{-\|\boldsymbol{v}_k\|\Delta t_k} & \text{otherwise,} \end{cases} \tag{21}$$

$$\boldsymbol{\mathcal{Y}}_k^{(i)}(y) = \begin{cases} \boldsymbol{\mathcal{Y}}_{k-1}^{(i)}(y) e^{-\|\boldsymbol{v}_k\|\Delta t_k} + 1 & \text{if } |y - y_k| < l, \\ \boldsymbol{\mathcal{Y}}_{k-1}^{(i)}(y) e^{-\|\boldsymbol{v}_k\|\Delta t_k} & \text{otherwise,} \end{cases} \tag{22}$$

where $x = 1, 2, ..., w$, and $y = 1, 2, ..., h$. and l is a tuning parameter controlling the size of the local neighborhood for the computation of the pixel's activity.

B. Generation of Endpoint Events

Pixels of a line are labeled as active if their activity is greater than a predefined threshold $\mathcal{A}_{up}^{(px)}$. The search for neighboring active pixels is then performed on the x or y projection, depending on the orientation of the line. Roughly speaking, if $|\theta_k^{(i)}| > \pi/4$ $\left(\text{or } \cos(\theta_k^{(i)}) < \cos(\pi/4)\right)$ we can say that the line is closer to being horizontal than vertical: in that case, each pixel of the line corresponds to a unique x coordinate, and we consequently perform our search on $\boldsymbol{\mathcal{X}}_k^{(i)}$. Otherwise we employ $\boldsymbol{\mathcal{Y}}_k^{(i)}$. This idea is illustrated in Fig. 3.

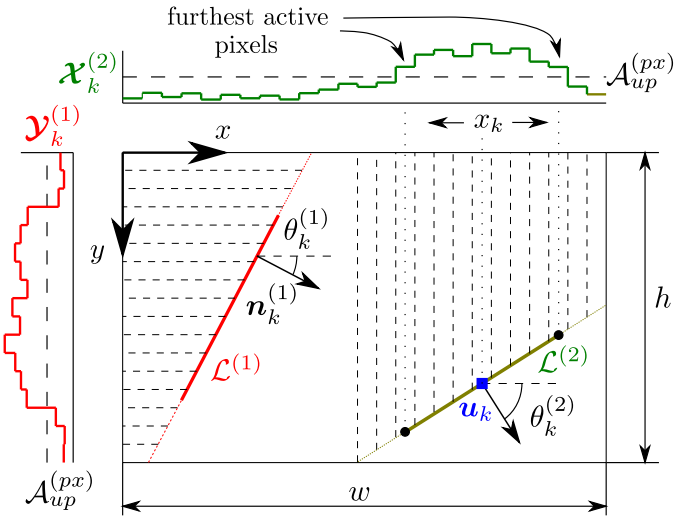

Fig. 3. The normal to $\mathcal{L}^{(1)}$, denoted $\boldsymbol{n}_k^{(1)}$, forms a small angle with the horizontal: $|\theta_k^{(1)}| < \pi/4$. We can then say that this line is closer to being vertical than horizontal, and each pixel of the line corresponds to a unique y position. Consequently, when an event is assigned to this line we perform our search for neighboring active pixels on this axis. The opposite holds true for $\mathcal{L}^{(2)}$: if the event $\mathbf{e}_k$ at position $\boldsymbol{u}_k$ is assigned to $\mathcal{L}^{(2)}$, we look for neighboring active pixels on $\boldsymbol{\mathcal{X}}_k^{(2)}$ starting at x_k. The furthest active pixel in each direction gives us an endpoint, and we generate two endpoint events at their positions.

When an oriented event $\mathbf{o}_k$ is assigned to a line $\mathcal{L}^{(i)}$ we look for active pixels in its neighborhood, starting at the position of the event. Thus, if $|\theta_k^{(i)}| > \pi/4$ we look for recent pixels in $\boldsymbol{\mathcal{X}}_k^{(i)}$ starting at x_k. Otherwise we perform our search in $\boldsymbol{\mathcal{Y}}_y^{(i)}$ starting at y_k. The endpoints are then given by the furthest active pixel in each direction, as shown in Fig. 3, where we impose a minimum length of $l/2$ to the segments. Two *endpoint events* are finally generated, giving the endpoints positions in space and time after a smoothing process using a blob tracker similar to the one introduced in [30].

C. Event-Based Segment Detection Algorithm

The event-based segment detection procedure is summarized in the Algorithm 2.

Algorithm 2 Event-Based Segment Detection

Require: $\mathbf{o}_k = [\boldsymbol{u}_k^T, \boldsymbol{v}_k^T, t_k, p_k]^T$ assigned to $\mathcal{L}^{(i)}$

Ensure: stream of endpoint events

for every oriented event $\mathbf{o}_k$ assigned to $\mathcal{L}^{(i)}$ **do**
  Update $\boldsymbol{\mathcal{X}}_k^{(i)}$ and $\boldsymbol{\mathcal{Y}}_k^{(i)}$ using (21) and (22)
  if $\mathcal{A}_k^{(i)} > \mathcal{A}_{up}^{(\mathcal{L})}$ **then**
    if $\cos\theta_k^{(i)} > \cos\frac{\pi}{4}$ **then**
      $x_1 \leftarrow x_k + 1$
      while $x_1 < w$ **and** $\boldsymbol{\mathcal{X}}_k^{(i)}[x_1] > \mathcal{A}_{up}^{(px)}$ **do**
        $x_1 \leftarrow x_1 + 1$
      end while
      $x_2 \leftarrow x_k - 1$
      while $x_2 \geq 0$ **and** $\boldsymbol{\mathcal{X}}_k^{(i)}[x_2] > \mathcal{A}_{up}^{(px)}$ **do**
        $x_2 \leftarrow x_2 - 1$
      end while
      if $x_1 - x_2 > l/2$ **then**
        $y_1 \leftarrow (x_1 \cos\theta_k^{(i)} - \rho_k^{(i)}) / \sin\theta_k^{(i)}$
        Generate an endpoint event $[x_1, y_1, t_k, p_k]^T$
        $y_2 \leftarrow (x_2 \cos\theta_k^{(i)} - \rho_k^{(i)}) / \sin\theta_k^{(i)}$
        Generate an endpoint event $[x_2, y_2, t_k, p_k]^T$
      end if
    else
      $y_1 \leftarrow y_k + 1$
      while $y_1 < h$ **and** $\boldsymbol{\mathcal{Y}}_k^{(i)}[y_1] > \mathcal{A}_{up}^{(px)}$ **do**
        $y_1 \leftarrow y_1 + 1$
      end while
      $y_2 \leftarrow y_k - 1$
      while $y_2 \geq 0$ **and** $\boldsymbol{\mathcal{Y}}_k^{(i)}[y_2] > \mathcal{A}_{up}^{(px)}$ **do**
        $y_2 \leftarrow y_2 - 1$
      end while
      if $y_1 - y_2 > l/2$ **then**
        $x_1 \leftarrow (y_1 \sin\theta_k^{(i)} - \rho_k^{(i)}) / \cos\theta_k^{(i)}$
        Generate an endpoint event $[x_1, y_1, t_k, p_k]^T$
        $x_2 \leftarrow (y_2 \sin\theta_k^{(i)} - \rho_k^{(i)}) / \cos\theta_k^{(i)}$
        Generate an endpoint event $[x_2, y_2, t_k, p_k]^T$
      end if
    end if
  end if
end for

IV. Results

In this section, we present two experiments to assess the accuracy of the algorithm with scenes captured by an Asyn-

chronous Time-Based Image Sensor (ATIS) with 304×240 pixels resolution [5]. In the first one, we applied the line and segment detection to a controlled scene for which we have built a ground truth. In the second experiment, we used two real scenes while moving the sensor. The algorithm was implemented in C++ and tested in a standard computer running Debian Linux.

A. Controlled moving lines

In order to numerically evaluate the results of the algorithm we first apply it to a pattern composed of 13 lines with a range of orientation between 0° and 90° at 7.5° intervals, as shown in Fig. 4A. The horizontal motion of the pattern is precisely controlled for different speeds. The scene is recorded using the event based camera as shown in Fig. 4B. The pattern is moved back and forth eight times for each speed.

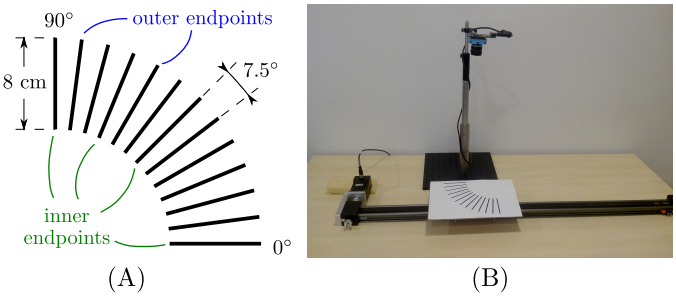

Fig. 4. (A) Controlled moving pattern composed of 13 lines with a range of orientation between 0° and 90° at 7.5°. When referring to endpoints, we will distinguish the *outer endpoints* and the *inner endpoints*. (B) Experimental setup: A printed pattern is mounted on a horizontal rail, which can be moved at different controlled speeds in the field of view of the event based ATIS camera, placed above.

The ground truth values for numerical evaluation of the algorithm are obtained from reconstructed frames, created by creating a frame every 1000 events. Since the movement of the object is always contained on a plane, its position w.r.t. the camera can be inferred by matching a planar template [32] estimated from a homographic transform. This is achieved with the Matlab implementation of the ECC algorithm [33] available online[1].

From these ground truth values we compute the true flow of the object, displayed in Fig. 5 (top). We verify that the observed velocity in the y axis remains approximately zero for the whole recording, except for some small vibrations. The x velocity takes much bigger values, going from positive to negative as the object moves back and forth. The maximum absolute value is equal to 654.2 px/s. As a comparison, in the first slowest movement the value of the flow in the x direction is around 60 px/s. Thus, the velocity in our scene varies by more than a factor of 10. Analogously, from these ground truth values we can obtain the apparent acceleration of the object on the image plane, denoted $[\alpha_x, \alpha_y]^T$ and expressed in px/s^2, which can be observed in Fig. 5 (bottom).

[1]https://fr.mathworks.com/matlabcentral/fileexchange/27253-ecc-image-alignment-algorithm–image-registration-

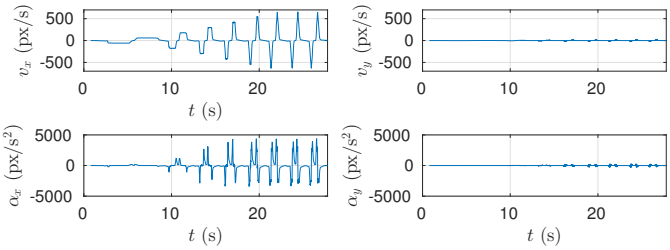

Fig. 5. Top: observed velocity of the object on the focal plane, in px/s. The velocity on the y axis remains close to zero for the whole recording, while the x velocity varies between negative and positive values, with a maximum absolute value of 654.2 px/s. Bottom: observed acceleration in px/s^2

1) Line detection: in the first place we evaluate just the line detection part of the algorithm, without performing segment detection. The tuning parameters used in this experiment are shown in the Table I given below.

TABLE I
LIST OF PARAMETERS FOR LINE DETECTION

d_{max} (px)	α_{max} (°)	$\mathcal{A}_{up}^{(\mathcal{L})}$	N
3	18	75	100

We show in Fig. 6A three snapshots depicting the recorded scene at three characteristic instants, where frames are reconstructed by plotting events happening within a 10 ms window. Fig. 6B shows the output of the normal flow algorithm at these same three instants, where the flow of each oriented event is represented by a straight line whose length is proportional to the norm of the flow. We verify that the flow is always normal to the lines, while its norm is smaller for the lines at smaller angles.

Fig. 6C shows the output of the line detection step at the same three instants, where active line models are superimposed on the events using a different color for each line index. In the leftmost snapshot we show the state of the system as the object starts its movement: we verify that all lines are detected, except the ones at 0° and 7.5°. This happens because the activity of these lines has not reached the threshold, for they are almost perpendicular to the direction of the movement and they consequently generate very few events. However, the norm of their flow is also smaller, which implies that their activity follows a slower decay: consequently, as the movement continues, the line at 7.5° reaches a sufficient level of activity and it is correctly tracked. The horizontal line, however, can seldom be detected. Let us remind the reader that the first step of our algorithm is the computation of the visual flow of the incoming events, which cannot be accomplished if the number of recent events in the neighborhood of the current event is not enough. A video showing the output of the algorithm for the whole sequence is included as supplementary material.

We display in Fig. 7 the evolution of the speed-tuned activity $\mathcal{A}_k$ for six different lines: the ones oriented at 90°, 75°, 60°, 45°, 30° and 15° (the rest of lines are not shown for simplicity, but equivalent results are obtained). We verify that the activity of the different lines is very similar, even though the ones with greater angles generate many more events. In the same way,

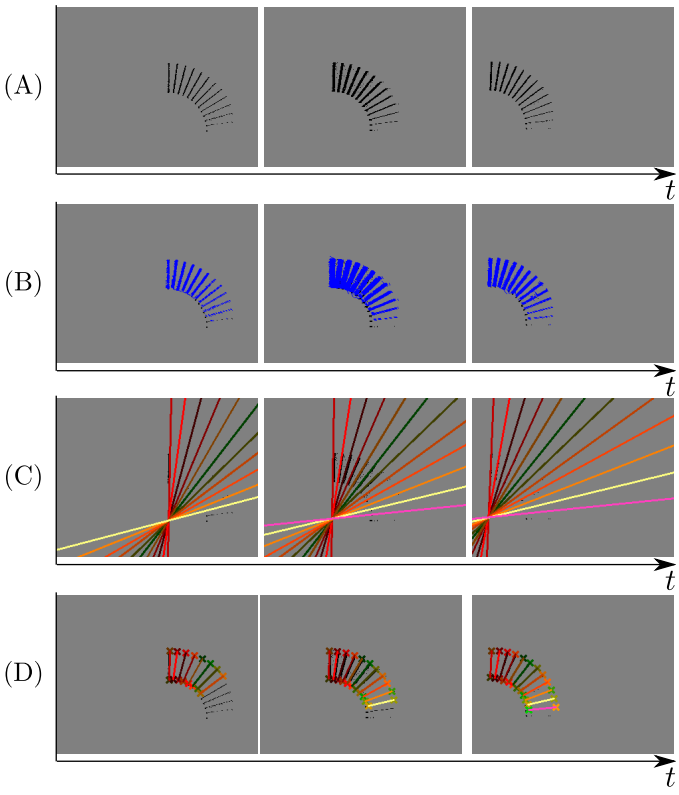

Fig. 6. (A) Frames reconstructed by plotting events happening within a 10 ms window, showing the recording at three characteristic instants. (B) Output of the normal flow algorithm: the flow is always normal to the local contours, while its norm is smaller for the lines at smaller angles. (C) Output of the line detection step. An index is assigned to each line as they are created, and we plot each line index with a different color. The horizontal line can seldom be tracked, as it generates very few events, preventing the computation of their flow. We are able of keeping track of the rest of the lines for the whole recording. (D) Output of the segment detection step. As the movement of the lines changes its direction we sometimes loose track of part of the endpoints, but we recover them again.

the activity is stable for the different velocities, which allows us to keep track of the lines for most of the recording.

Additionally, we compare the results of the speed-tuned decay activity to a constant decay one. The constant decay function is given by $e^{-\Delta t_k/\tau}$, with τ a tuning parameter. Here, we choose $\tau = 20000$ µs, which yields values for the fixed decay activity in the same order of magnitude as the speed-tuned one. The obtained values are displayed in Fig. 7: we verify that the fixed decay activity is very sensitive to the orientation and the apparent velocity of the lines, imposing the need of adjusting the tuning parameter τ according to the velocity and the direction of movement of the observed objects. We show in Fig. 8 the percentage of time for which the different lines are active when applying both the speed-tuned and the fixed decay strategies with the current set of parameters. As we can see, the speed-tuned strategy is more reliable as it yields higher percentages.

However, we can observe in Fig. 7 that the activity decreases during deceleration periods: this corresponds to an expected behavior, due to the adaption of the speed-tuned decay strategy (see Section II-C). This is particularly significant when the apparent motion is globally slower (in particular when looking at the activity curve for the line at 30°, Fig. 7), because

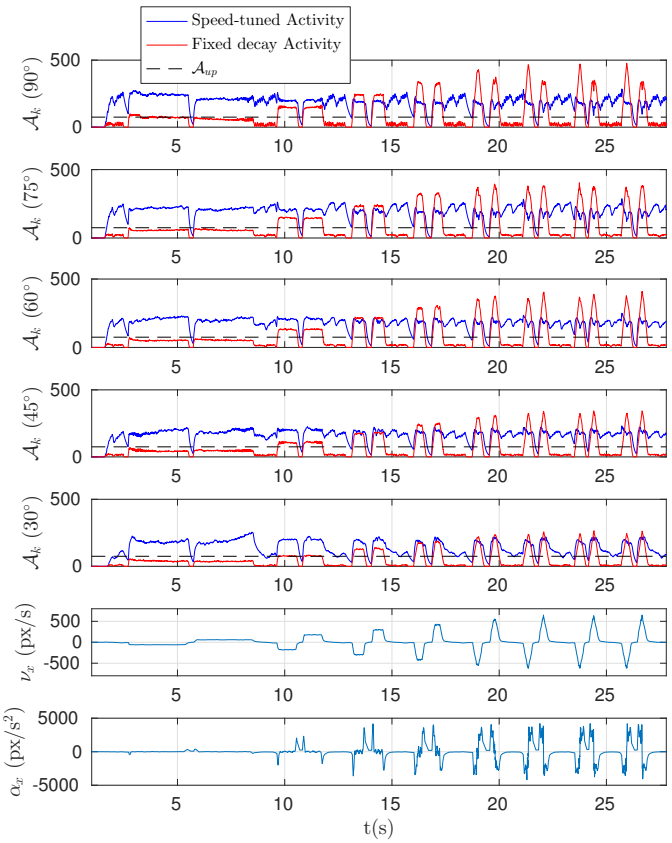

Fig. 7. Evolution of the activity for six different lines. We display the results obtained with both the speed-tuned decreasing strategy and the fixed decreasing strategy. We verify that the speed-tuned strategy is much more stable, in spite of the different orientations and apparent velocities. The dashed line indicates the threshold $\mathcal{A}_{up}$.

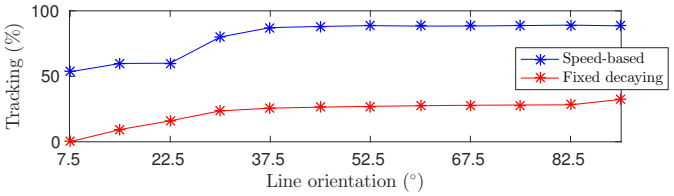

Fig. 8. Percentage of the recording for which we are able of keeping track of the lines, for both the speed-tuned and the fixed decay strategies. We verify that the speed-tuned method is more stable

the number of events generated by the silicon retina is then smaller. The algorithm is then more sensitive to error in velocity estimation and internal noise of the sensor's (mismatches, thermal noise, etc.). These observations are confirmed regarding the errors in model estimation.

Let us next numerically evaluate the error in the estimation of the line parameters. Here, we evaluate ρ and θ independently and we define two errors: ξ_ρ (in pixels) and ξ_θ (in degrees), given by the absolute value of the difference between the estimated and the ground truth values of the corresponding parameters. We show in Fig. 9 the obtained values for the lines oriented at 90°, 75°, 60°, 45°, 30° and 15°, where the error is computed only when the corresponding line is active. From this figure, we can extract the following conclusions:

- In general, the algorithm is capable of correctly estimat-

ing the parameters of the observed lines. The values of the errors remain small for all lines for the whole duration of the recording.

- Lines parallel to the motion are harder to detect and track since the motion is almost not triggering events. This hampers the computation of the optical flow, yielding less stable results.
- We loose lines at smaller angles more often. This usually happens when the lines stop their movement.

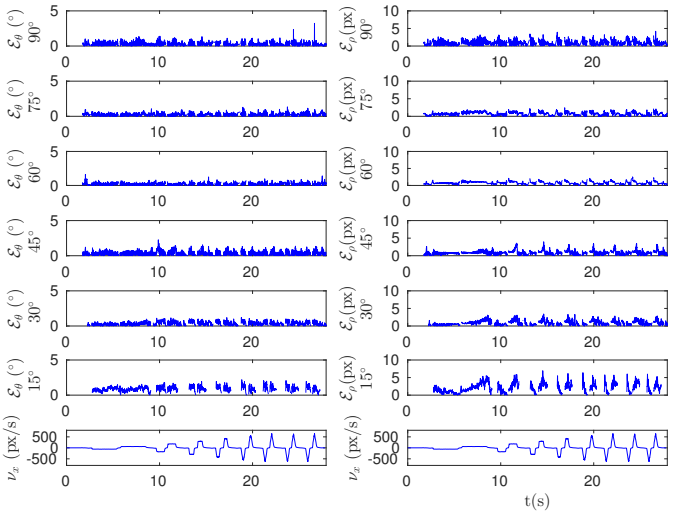

Fig. 9. Errors in the estimation of ρ and θ for six of the lines contained in the scene. Results for almost horizontal lines are less stable, but we are still able of correctly tracking these lines.

In order to establish a clearer comparison between the accuracy produced for the different orientations let us plot in Fig. 10 the values of the mean errors obtained for the different lines for the whole duration of the recording. We verify that these errors are bigger for smaller angles. As an example, for the line at $90°$ (which moves perpendicularly to its contour) we obtain mean errors of just $0.31°$ and 0.90 px.

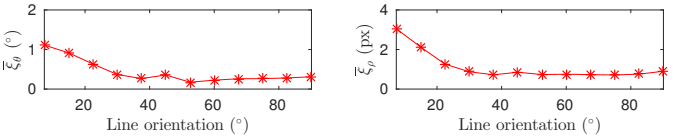

Fig. 10. Evolution of the mean error in the estimation of θ and ρ with the orientation of the line. We verify that the best results are obtained for lines oriented perpendicularly to the direction of the movement.

2) Segment detection: the parameters for the line detection are the same used in the previous experiment (indicated in Table I), while the extra parameters required for segment detection are: $l = 10$ px, $\mathcal{A}_{up}^{(px)} = 1.5$.

We show in Fig. 6D the output produced by the segment detection step, where the tracked endpoints are indicated by crosses in different colors for each endpoint. In the leftmost snapshot we observe the state of the detection as the object initiates its movement: we verify that some of the endpoints are not detected in this first instant, since their activity has not reached its threshold yet. Like in the previous experiment, as the object continues moving, we are able of detecting the missing endpoints. All endpoints, except the ones belonging to the horizontal line, are correctly detected and tracked for the whole duration of the recording. A video showing the sequence is included as supplementary material.

We next compare the obtained results with the ground truth values. We plot in Fig. 11 the tracking results obtained for three endpoints, namely the outer endpoints of the lines at $90°$, $45°$ and $15°$. In the figure, ground truth values are indicated with a dashed line and compared with the obtained results. As we can see, they are very similar, allowing us to conclude that our method can correctly detect the endpoints in this simple scene.

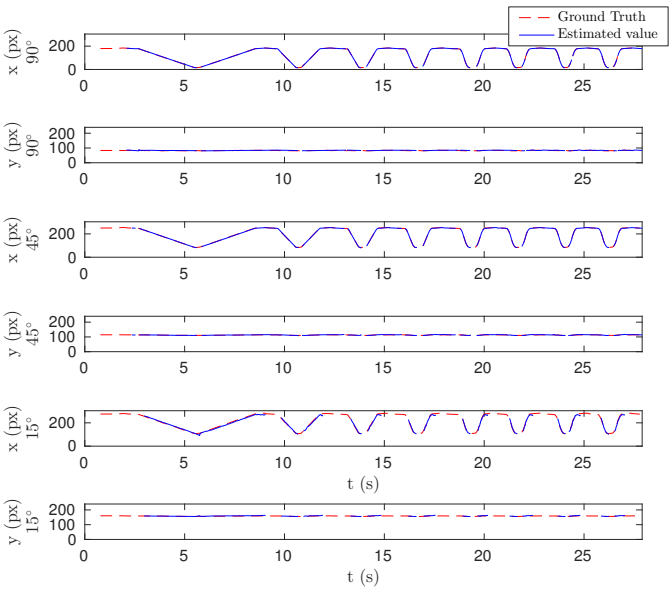

Fig. 11. Tracking results for the outer endpoints of the lines at $90°$, $45°$ and $15°$.

We provide also a quantitative evaluation of the tracking by showing the tracking error ξ_t (in pixels) as the distance between the tracked endpoints and the corresponding ground truth values. We only take into account the instants when the endpoint trackers are active. Consequently, we will also evaluate the percentage of the recording for which the endpoints are detected. We show in Fig. 12 the mean tracking errors produced over the whole recording for the different endpoints. As we can see, errors are greater for smaller angles. As an example, the mean tracking errors committed for the endpoints belonging to the vertical line are 1.49 px for the outer endpoint, and 1.54 px for the inner endpoint.

In the same way, the smaller the angle the smaller the percentage of the time that we can track the endpoints. We are able of tracking the endpoints in the vertical line for 84.6% of the time.

To conclude about endpoint detection, their spatial imprecision around 1 pixel can be explained by the attribution of the events to the lines, related to the matching between the visual flow computed around them and the line model (see Section II.B.3.). The proposed computation of the optical flow is indeed less robust at the endpoints, due to the weaker number of events (or to events belonging to other edges) in the spatiotemporal neighborhood of ending points than around

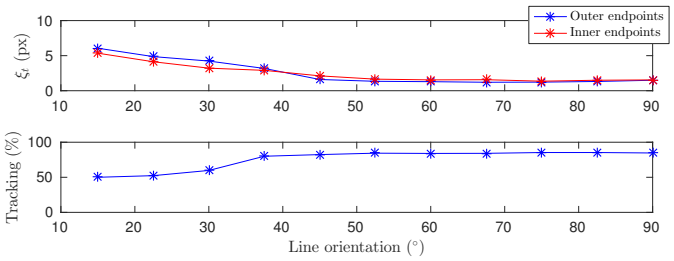

Fig. 12. Accuracy of the endpoint tracking for the different endpoints. We obtain more accurate results and greater percentage of active time for lines with greater angles.

other points along the edges. Thus, events corresponding to endpoints are sometimes not attributed to the line, which implies a weaker local activity, and consequently the imprecision of the endpoints' locations. About robustness, essentially for the same reasons, the algorithm fails when the apparent motion is too weak (and the number of attributed events is therefore also weak).

Despite this extreme condition, we have proven that the algorithm behaves similarly whatever the speed or the orientation of lines.

B. Real scenes

We next show the output of the algorithm applied to two real scenes. Both recordings were performed with a hand-held ATIS camera moving in circles. We process them using the same parameters as in the previous case, except for the maximum number of line models, which is increased until $N = 1000$. In this case, we do not provide any quantitative measurement of the accuracy produced, as ground truth values are not easily attainable. Instead, these results are just shown as a qualitative demonstration of what can be achieved with our algorithm (videos of each experiment are included as supplementary material). This is also a key opportunity to discuss some properties of the proposed method, regarding its potential applications.

1) Urban environment: we first apply the method to a recording of an urban landscape, which contains a large number of lines. We show in Fig. 13A three snapshots reconstructed from the recording, while Fig. 13B depicts the output of the line detection step. We observe that horizontal or vertical lines tend to predominate depending on the direction of the movement at the corresponding instant.

In Fig. 13C the output of the segment detection is also shown. For visibility reasons, we do not display any crosses marking the endpoints' positions. We verify that our algorithm is capable of recovering an important part of the segments present in the recorded scene. A video showing the output of both the line detection and segment detection steps is included as supplementary material.

2) Office: the second real scene tested with the detection/tracking algorithm is the recording of an office. Fig. 14A shows three snapshots obtained from the recording: we observe that the scene contains a great number of lines, whose lengths are smaller than in the urban scene recording. As we can see in Fig. 14B, many of these shorts segments are not detected

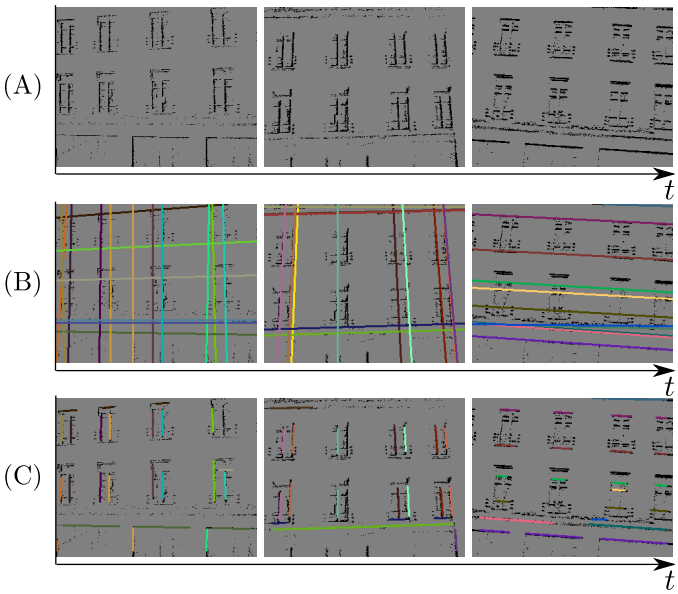

Fig. 13. (A) Frames reconstructed from the recording. (B) Output of the line detection step. (C) Output of the segment detection step.

because their activities are not normalized with respect to their lengths, which causes the longer segments to be more likely to be detected.

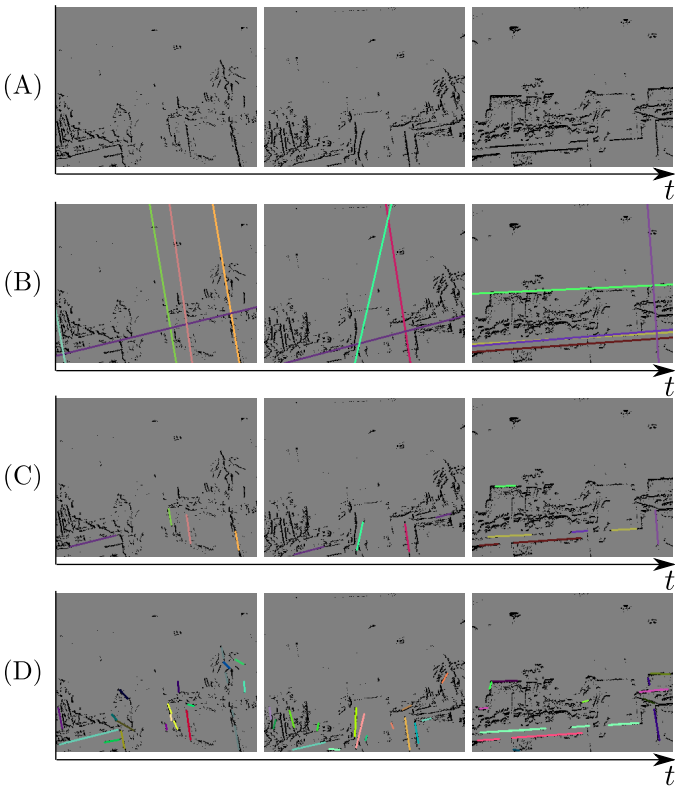

Fig. 14. (A) Frames reconstructed from the recording. (B) Output of the line detection step of the algorithm. (C) Output of the segment detection part of the algorithm. (D) Output of the segment detection part of the algorithm after tweaking the parameters.

We show in Fig. 14C the output of the segment detection algorithm when processing this scene with the same set of parameters as for the urban environment. We verify

that many of the small segments are not detected. Indeed, the proposed method tends to emphasize long and highly contrasted contours because they generates more events. This characteristic, common to most segment detection algorithms (frame or event-based), is often required when considering the motion of a camera relative to its environment (as it is the case in robotics and navigation applications). In this type of visual scenes, long and contrasted segments provide robust information, as high contrast and long size make them visible for long periods of time, even if the global illumination and/or the point of view vary.

It is also possible to adjust the parameters, reducing the activity threshold $\mathcal{A}_{up}^{(\mathcal{L})}$ and the length of the window l. We show in Fig. 14D the output produced after tuning the parameters: as we can see, we are able of detecting a greater number of small segments. Of course, the algorithm's parameters can be automatically adjusted, requiring a given number of lines and a maximum number of segments allowed per line.

C. Computational time

Let us next evaluate the computational time required by our current C++ implementation of the algorithm. These tests were performed in a standard computer running Debian Linux, equipped with a Intel Core i7-4790 processor. The code was not parallelized and just one core was used.

In order to characterize the computational time required by the algorithm we measure the time it takes to process the previously presented recordings. To that end, we measure the processing time for every time period of one millisecond (without overlapping) and compute the ratio of processing time to the length of the considered periods (i.e. the number of milliseconds it takes to process one millisecond of events). Thus, if this ratio is smaller than 1, the algorithm can process the corresponding event stream online without increasing latency (i.e. in "real-time video").

In order to obtain stable values we process each recording 10 times and average the obtained results.

Let us show in Fig. 15 the ratio of processing time to the length of the recording, obtained when processing the first simple scene see Section IV-A2 and Figures 4 and 6) with the set of parameters given in Table I. Let us note that we are considering the processing time required by the whole processing chain: this includes reading the recording file, preprocessing the recording for noise removal and computing the visual flow of the incoming events, in addition to the line (or segment) detection. We display at the top of Fig. 15 the computational time required to perform line detection, while the results for segment detection are shown at the bottom.

From Fig. 15 we can observe:

- The ratio of processing time to the length of the recording is always smaller than 1. This implies that we are processing the event stream faster than we acquire it (both for segment and line detection).
- The segment detection algorithm is around 10 percent more computationally demanding, on average
- This ratio is clearly correlated to the speed of the object (see Fig. 5). This can be easily explained because faster moving objects generate a bigger number of events that need to be processed.

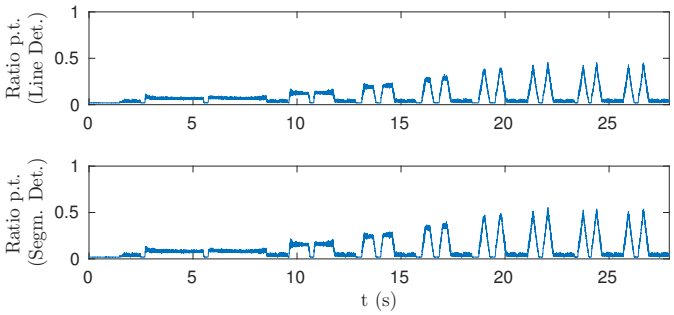

Fig. 15. We measure the processing time every time periods of 1 ms and compute the ratio of processing time to the length of the considered periods. We show the results obtained when processing the first recording (simple scene) for both line and segment detections. Because this ratio is always smaller than 1, we can conclude that we are processing this event stream faster than we acquire it (in "real-time video"). The computational time is related to the speed of the object, because faster moving objects generate greater number of events.

According to this last point, it is more interesting to display the ratio of processing time as a function of the event rate, as in [34]. Let us next study the computational time required to process the urban environment recording, which contains a greater number of events. We show in Fig. 16 the ratio of processing time for three different values of the parameter N (i.e. the maximum number of lines that can be initialized). We verify that the computational time is increasing with the number of events, and can be approximated by a linear function. This allows us to extrapolate and compute the maximum rate of events that can be processed online (i.e. faster than we acquire them) by our algorithm, depending on the value of N.

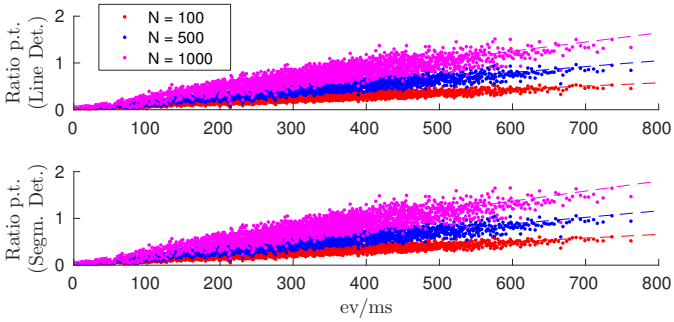

Fig. 16. Computational time per ms (for line detection -top Figure- and segment detection -bottom Figure) as a function of the number of events per ms. The required computational time increases with the event rate: regressions of these data are represented as dashed lines in order to highlight their linear dependency, related to the parameter N (i.e. the number of considered line models).

We then show in Fig. 17 the evolution of the maximum event rate that can be processed online by the algorithm, both for line detection (top) and segment detection (bottom). Indeed, we verify that, even for big values of N, we are able to process big event rates in "real-time video" As an example, for $N = 500$ we can perform online line detection for event rates up to 760 ev/ms, and online segment detection for event rates up to 684 ev/ms.

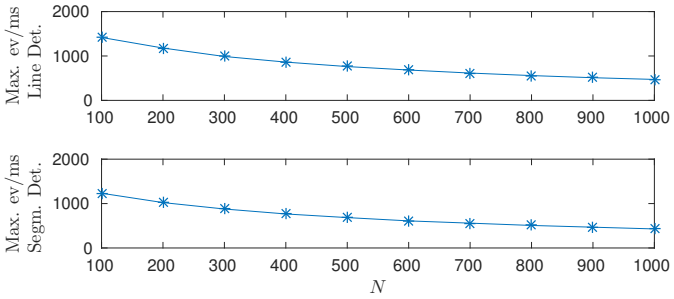

Fig. 17. Maximum event rate that can be processed online, as a function of the parameter N (Top: line detection. Bottom: segment detection).

Note that, if bigger event rates are required for a given application, a parallelization of the proposed algorithm can be easily envisioned: for example, the updating of the N model lines (and segments) could be shared by several different cores.

V. Conclusion

We have presented a new event-based line and segment detection algorithm. Based on an iterative process, our approach is fully event-based, as each event is used to update the current model and forgotten afterwards. This latter is obtained by applying a speed-tuned exponential decay to the contribution of each event to the line model, and then to the segment model. Our approach takes into account the dynamics of the visual information through a local computation of the optical flow, ensuring that the models are essentially estimated on the current visual information. The proposed line detection is based on a iterative least squares fitting scheme that is compatible with the asynchronous data.

These properties are validated through an experimental protocol, involving a simple scene for which ground truth values are available. This allows us to assess the accuracy and the robustness of the algorithm in a controlled environment. Indeed, the mean accuracy obtained for the line model parameters, i.e. the distance to the origin and the angle, are in most cases lower than 1 pixel and 1° respectively. The results for the segment endpoints are also accurate: for example, the mean tracking errors committed for a line perpendicular to the motion are around 1.5 pixels. In addition, a complete analysis of the computational time required by the proposed algorithms is proposed. This analysis shows that our current implementation is able to process online event streams. The method could be easily parallelized in further developments for demanding applications.

These characteristics make our event-based line and segment detection algorithms suitable for real-time applications, such as navigation of robotic platforms in real environments (particularly for line-based SLAM algorithms [14]-[16]). Due to their event-based processing fully exploiting the neuromorphic cameras' quasi-continuous acquisition of visual information, the proposed methods make particularly a fitting for high-speed applications, because they are not constrained by sampled acquisition as in frame-based vision.

Acknowledgments

Funding This work received financial support from the LABEX LIFESENSES [ANR-10-LABX-65] which is managed by the French state funds (ANR) within the Investissements d'Avenir program [ANR-11-IDEX-0004-02]. The robotic platform used in the experimental section has been funded by the European Union Seventh Framework Program, in the context of a sub-task (named asynchronous COmputational REtina; aCORE) of the Human Brain Project, grant agreement number 604102.

Appendix A
Optimization strategy

Let us rewrite (13) dropping the superindices, in order to lighten the notation:

$$a_k \sin(2\theta_k) + b_k \cos(2\theta_k) = 0. \tag{23}$$

From this we obtain:

$$\pm a_k \sqrt{1 - \cos^2(2\theta_k)} = -b_k \cos(2\theta_k),$$

which yields:

$$a_k^2\big(1-\cos^2(2\theta_k)\big) = b_k^2 \cos^2(2\theta_k) \;\Rightarrow\; \cos^2(2\theta_k) = \frac{a_k^2}{a_k^2 + b_k^2}.$$

From this we get:

$$\cos(2\theta_k) = \beta_k, \tag{24}$$

where $\beta_k = \pm\sqrt{\frac{a_k^2}{a_k^2+b_k^2}}$.

Next, we insert into (24) the following trigonometric identities: $\cos(2\theta) = 2\cos^2(\theta) - 1$, and $\cos(2\theta) = 1 - 2\sin^2(\theta)$. We obtain:

$$\cos(\theta_k) = \sqrt{\frac{\beta_k + 1}{2}}, \quad \sin(\theta_k) = \pm\sqrt{\frac{1 - \beta_k}{2}}. \tag{25}$$

Where we only keep the positive sign for the cosinus, because $\theta_k \in [-\pi/2,\ \pi/2]$.

Appendix B
Disambiguation

Let us define γ_1, γ_2:

$$\gamma_1 = \sqrt{\frac{1 - |\beta_k|}{2}}, \quad \gamma_2 = \sqrt{\frac{1 + |\beta_k|}{2}}, \tag{26}$$

From (19) we obtain four possible combinations for $\cos\theta_k$, $\sin\theta_k$ (where we drop the superindex indicating the line in order to lighten the notation):

$$\begin{cases} \text{if } \beta_k = +|\beta_k| \;\Rightarrow\; \sin(\theta_k) = \pm\gamma_1, \quad \cos(\theta_k) = \gamma_2 \\ \text{if } \beta_k = -|\beta_k| \;\Rightarrow \cos(\theta_k) = \gamma_1, \quad \sin(\theta_k) = \pm\gamma_2 \end{cases} \tag{27}$$

Next, we need to disambiguate for the sign of the sinus. Here, we take into account that $\tan 2\theta_k = \frac{-b_k}{a_k}$. Then, as one can see in Fig. 18, the following rule applies:

$$\begin{cases} \text{if } \dfrac{-b_k}{a_k} > 0 \begin{cases} \text{if } \cos(\theta_k) < \cos(\pi/4) \;\Rightarrow\; \sin(\theta_k) < 0 \\ \text{if } \cos(\theta_k) > \cos(\pi/4) \;\Rightarrow\; \sin(\theta_k) > 0 \end{cases} \\ \text{if } \dfrac{-b_k}{a_k} < 0 \begin{cases} \text{if } \cos(\theta_k) < \cos(\pi/4) \;\Rightarrow\; \sin(\theta_k) > 0 \\ \text{if } \cos(\theta_k) > \cos(\pi/4) \;\Rightarrow\; \sin(\theta_k) < 0 \end{cases} \end{cases} \tag{28}$$

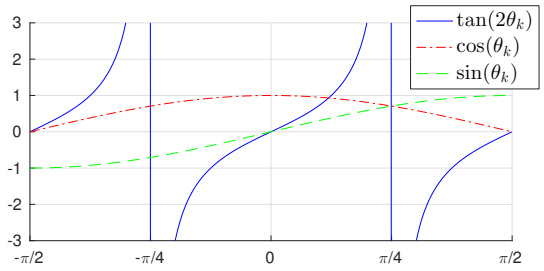

Fig. 18. The sign of the sinus can be disambiguated from the value of λ and $\cos(\theta_k)$.

This leaves us with two possible combinations, which correspond to the two perpendicular lines yielding the maximum and the minimum error. We disambiguate between then by choosing the value of $\cos(\theta_k)$ closest to the previous one. Here, we choose to compare the cosinus because for almost horizontal lines we have $\theta_k \approx \pm\pi/2$. Values of θ_k close to $\pi/2$ or $-\pi/2$ will have a similar cosinus, allowing for the line to change from one to another.

Alternatively, it is possible to disambiguate between the two combinations by imposing the second derivative to be greater than zero. This yields the following condition:

$$\begin{aligned} 0 < \cos(2\theta_k)\left(\widehat{yy}_k - \widehat{xx}_k\right) - 2\sin(2\theta_k)\widehat{xy}_k \\ + \rho_k\left(\widehat{x}_k\cos(\theta_k) + \widehat{y}_k\sin(\theta_k)\right). \end{aligned} \tag{29}$$

In our implementation we choose to select the value closest to the previous one, as this method is less computationally demanding.

REFERENCES

[1] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbrück, "Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output," *Proc. of the IEEE*, vol. 102, no. 10, pp. 1470–1484, Oct 2014.

[2] M. Mahowald, "VLSI analogs of neuronal visual processing: a synthesis of form and function," Ph.D. dissertation, California Institute of Technology, 1992.

[3] T. Delbrück, "Silicon retina with correlation-based, velocity-tuned pixels," *IEEE Trans. on Neural Networks*, vol. 4, no. 3, pp. 529–541, May 1993.

[4] P. Lichtsteiner, C. Posch, and T. Delbrück, "A 128 128× 128 120 dB 15µs latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb 2008.

[5] C. Posch, D. Matolin, and R. Wohlgenannt, "An asynchronous time-based image sensor," in *Proceedings of the IEEE International Symposium on Circuits and Systems*. Seattle, WA, USA: IEEE, May 2008, pp. 2130 –2133.

[6] ——, "A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, Jan 2011.

[7] R. Etienne-Cummings, J. Van der Spiegel, and P. Mueller, "A focal plane visual motion measurement sensor," *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol. 44, no. 1, pp. 55–66, Jan 1997.

[8] J. Krammer and C. Koch, "Pulse-based analog VLSI velocity sensors," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 44, no. 2, pp. 86–101, Feb 1997.

[9] T. Delbrück, B. Linares-Barranco, E. Culurciello, and C. Posch, "Activity-driven, event-based vision sensors," in *Proc. of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. Paris, France: IEEE, May 2010, pp. 2426–2429.

[10] C. Bartolozzi, R. Benosman, K. Boahen, G. Cauwenberghs, T. Delbrück, G. Indiveri, S.-C. Liu, S. Furber, N. Imam, B. Linares-Barranco, T. Serrano-Gotarredona, K. Meier, C. Posch, and M. Valle, *Neuromorphic Systems*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2016.

[11] Y. Lu and D. Song, "Robust rgb-d odometry using point and line features," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3934–3942.

[12] G. Schindler, P. Krishnamurthy, and F. Dellaert, "Line-based structure from motion for urban environments," in *3D Data Processing, Visualization, and Transmission, Third International Symposium on*. IEEE, 2006, pp. 846–853.

[13] L. Zhang and R. Koch, "Structure and motion from line correspondences: representation, projection, initialization and sparse bundle adjustment," *Journal of Visual Communication and Image Representation*, vol. 25, no. 5, pp. 904–915, 2014.

[14] W. Y. Jeong and K. M. Lee, "Visual slam with line and corner features," in *Intelligent Robots and Systems, 2006 IEEE/RSJ Int. Conf. on*. IEEE, 2006, pp. 2570–2575.

[15] T. Lemaire and S. Lacroix, "Monocular-vision based slam using line segments," in *Robotics and Automation, 2007 IEEE Int. Conf. on*. IEEE, 2007, pp. 2791–2796.

[16] J. Lv, Y. Kobayashi, A. A. Ravankar, and T. Emaru, "Straight line segments extraction and ekf-slam in indoor environment," *Journal of Automation and Control Engineering*, vol. 2, no. 3, 2014.

[17] J. Illingworth and J. Kittler, "A survey of the Hough transform," *Computer vision, graphics, and image processing*, vol. 44, no. 1, pp. 87–116, 1988.

[18] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[19] P. Mukhopadhyay and B. B. Chaudhuri, "A survey of Hough transform," *Pattern Recognition*, vol. 48, no. 3, pp. 993–1010, 2015.

[20] J. B. Burns, A. R. Hanson, and E. M. Riseman, "Extracting straight lines," *IEEE trans. on pattern analysis and machine intelligence*, no. 4, pp. 425–455, 1986.

[21] A.-R. Mansouri, A. S. Malowany, and M. D. Levine, "Line detection in digital pictures: A hypothesis prediction/verification paradigm," *Computer Vision, Graphics, and Image Processing*, vol. 40, no. 1, pp. 95–114, 1987.

[22] v. G. R. Grompone, J. Jakubowicz, J.-M. Morel, and G. Randall, "LSD: a fast line segment detector with a false detection control." *IEEE trans. on pattern analysis and machine intelligence*, vol. 32, no. 4, pp. 722–732, 2010.

[23] C. Brandli, J. Strubel, S. Keller, D. Scaramuzza, and T. Delbrück, "ELiSeD - an event-based line segment detector," in *2016 Second Int. Conf. on Event-based Control, Communication, and Signal Processing (EBCCSP)*, June 2016, pp. 1–7.

[24] D. R. Valeiras, S. Kime, S.-H. Ieng, and R. B. Benosman, "An event-based solution to the perspective-n-point problem," *Frontiers in neuroscience*, vol. 10, 2016.

[25] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow," *IEEE trans. on neural networks and learning systems*, vol. 25, no. 2, pp. 407–417, 2014.

[26] Å. Björck, *Numerical methods in matrix computations*. Berlin, Germany: Springer, 2015.

[27] L.-W. Chang, W.-N. Lie, and R. Chiang, *Advances in Image and Video Technology: First Pacific Rim Symposium, PSIVT 2006, Hsinchu, Taiwan, December 10-13, 2006, Proceedings*. Springer, 2006, vol. 4319.

[28] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[29] R. Larson and B. H. Edwards, *Calculus of a single variable*. Cengage Learning, 2013.

[30] 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. on Neural Networks and Learning Systems*, vol. PP, p. 1, Sep. 2014.

[31] X. Clady, J.-M. Maro, S. Barré, and R. Benosman, "A motion-based feature for event-based pattern recognition," *Frontiers in Neuroscience*, vol. 10, p. 594, 2016.

[32] R. Brunelli, *Template matching techniques in computer vision: theory and practice*. Hoboken, NJ, USA: John Wiley & Sons, 2009.

[33] G. D. Evangelidis and E. Z. Psarakis, "Parametric image alignment using enhanced correlation coefficient maximization," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 30, no. 10, pp. 1858–1865, 2008.

[34] C. Simon-Chane, S.-H. Ieng, C. Posch, and R. B. Benosman, "Event-based tone mapping for asynchronous time-based image sensor," *Frontiers in Neuroscience*, vol. 10, 2016.