Asynchronous Event-Based Multikernel Algorithm for High-Speed Visual Features Tracking

Xavier Lagorce, Cédric Meyer, Sio-Hoi Ieng, David Filliat, and Ryad Benosman

***Abstract*— This paper presents a number of new methods for visual tracking using the output of an event-based asynchronous neuromorphic dynamic vision sensor. It allows the tracking of multiple visual features in real time, achieving an update rate of several hundred kilohertz on a standard desktop PC. The approach has been specially adapted to take advantage of the event-driven properties of these sensors by combining both spatial and temporal correlations of events in an asynchronous iterative framework. Various kernels, such as Gaussian, Gabor, combinations of Gabor functions, and arbitrary user-defined kernels, are used to track features from incoming events. The trackers described in this paper are capable of handling variations in position, scale, and orientation through the use of multiple pools of trackers. This approach avoids the N^2 operations per event associated with conventional kernel-based convolution operations with $N \times N$ kernels. The tracking performance was evaluated experimentally for each type of kernel in order to demonstrate the robustness of the proposed solution.**

***Index Terms*— Event-based vision, neuromorphic sensing, visual tracking.**

I. Introduction

VISUAL object recognition and tracking is useful in many applications such as video surveillance, traffic monitoring, motion analysis, augmented reality, and autonomous robotics. Most object tracking techniques rely on sequences of static frames, which limits algorithmic efficiency when dealing with highly dynamic scenes. Conventional frame-based video cameras can acquire data at frequency as high as several tens of kilohertz, nevertheless this amount of data remains difficult to process in real time due to the large amount of redundant acquired information. Real-time processing at high acquisition rates usually requires different techniques such as: 1) subsampling of the field-of-view [1]; 2) the use of specific hardware implementations; or 3) a restriction to simple tracking algorithm such as image's centroid and moments computation [2].

This paper presents an event-based approach to fast visual tracking of features using the output of an asynchronous neuromorphic event-based camera. Neuromorphic cameras (sometimes called silicon retinas) mimic the biological visual systems [3], [4]. The asynchronous time-based image sensor (ATIS) camera used in this paper reacts to changes of scene contrast and records only dynamic information, thus reducing the data quantity [5], [6]. The presented algorithms rely on the change detection circuit of the ATIS, namely it only deals with relative change events. It can also be applied to other sensors, such as the dynamic vision sensor (DVS) [4], on which the ATIS is based.

The presented tracking algorithm is computationally inexpensive and is thus capable of tracking objects and updating their properties at rates in the order of hundreds of kilohertz. Initially, an asynchronous event-based Gaussian blob tracking algorithm is developed and examined. The properties of the Gaussian kernel allow adaptation to the events' spatial distribution by continuously correcting the Gaussian size, orientation, and location with each incoming event. This provides the object's position, size, and orientation simultaneously. In a second stage, the model is extended using oriented Gabor kernels that allow tracking specific-oriented edges. This paper also considers the combination of several oriented kernels that are useful in tracking specific focal plane structures such as corners. Finally, a general kernel approach is presented. It can use almost any arbitrary kernel and can be seen as a generalization of the process, with the only constraint that their center of mass has to be aligned with the center of the kernels (Section III-C4).

II. Time Encoded Imaging

Biomimetic event-based cameras are a novel type of vision devices that—like their biological counterparts—are driven by events happening within the scene. They are not like conventional vision sensors, which, by contrast, are driven by artificially created timing and control signals (e.g., frame clock), which have no relation whatsoever to the source of the visual information [7]. Over the past few years, a variety of these event-based devices have been implemented, including temporal contrast vision sensors that are sensitive to relative luminance change, gradient-based sensors sensitive to static edges, edge-orientation sensitive devices, and optical-flow sensors. Most of these vision sensors output visual information about the scene in the form of asynchronous address events [8]

Manuscript received April 2, 2013; revised April 22, 2014, August 5, 2014, and August 12, 2014; accepted August 19, 2014. Date of publication September 16, 2014; date of current version July 15, 2015.

X. Lagorce, S.-H. Ieng, and R. Benosman are with the Vision and Natural Computation Group, Institut National de la Santé et de la Recherche Médicale, Paris F-75012, France, Sorbonne Universités, Institut de la Vision, Université Paris 06, Paris F-75012, France, and also with Centre National de la Recherche Scientifique, Paris F-75012, France (e-mail: xavier.lagorce@ens-cachan.fr; cedric.meyer@ens-cachan.fr; sio-hoi.ieng@upmc.fr; ryad.benosman@upmc.fr).

C. Meyer is with the University Institute of Technology of Angouleme, Angouleme F-16000, France (e-mail: cedric.meyer@univ-poitiers.fr).

D. Filliat is with École Nationale Supérieure de Techniques Avancées, Paris F-75015, France (e-mail: david.filliat@ensta.fr).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org (File size: 12 MB).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2014.2352401

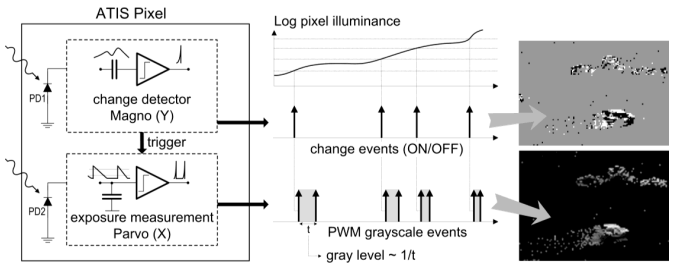


Fig. 1. Functional diagram of an ATIS pixel [5]. Two types of asynchronous events, encoding change, and brightness information are generated and transmitted individually by each pixel in the imaging array.

and encode the visual information in the time dimension and not as voltage, charge, or current. The presented pattern tracking method is designed to work on the data delivered by such a time-encoding sensor and takes full advantage of the high temporal resolution and sparse data representation.

The ATIS used in this paper is a time-domain encoding vision sensors with 304×240 pixels resolution [6]. The sensor contains an array of fully autonomous pixels that combine an illuminance relative change detector circuit and a conditional exposure measurement block.

As shown in the functional diagram of the ATIS pixel in Fig. 1, the relative change detector individually and asynchronously initiates the measurement of an exposure/gray scale value only if—and immediately after—a brightness change of a certain magnitude has been detected in the field-of-view of the respective pixel. The exposure measurement circuit in each pixel individually encodes the absolute instantaneous pixel illuminance into the timing of asynchronous event pulses, represented as interevent intervals.

Since the ATIS is not clocked like a conventional camera, the timing of events can be conveyed with a very accurate temporal resolution in the order of microseconds. The time-domain encoding of the intensity information automatically optimizes the exposure time separately for each pixel instead of imposing a fixed integration time for the entire array, resulting in an exceptionally high-dynamic range and improved signal-to-noise ratio. The pixelwise change detector-driven operation yields almost ideal temporal redundancy suppression, resulting in a maximally sparse encoding of the image data.

In what follows, we will rely only on change detector events as the timings of events are the main needed information to perform tracking.

III. Multikernel Event-Based Features Tracking

This section describes the tracking algorithms developed in this paper. After reviewing the state of the art of asynchronous tracking using event-based silicon retinas, we describe how bivariate normal distributions are used to track clouds of events. Then, we will generalize our approach to more arbitrary kernels. The following two sections tackle the problem of tracking several objects at once in a scene and how several trackers interact with one another in a pool mechanism. To conclude this section, a global algorithm is presented and a number of possible optimizations are proposed.

A. State of the Art

Image segmentation, feature extraction, optical flow, and high-level motion filters are usually used to track moving objects but they are known to be computationally expensive tasks. Real-time processing with frame rates reaching several hundreds of hertz can lead to more robust tracking algorithms [9]. However, it is currently an almost impossible task to perform such tasks in real time unless a dedicated hardware is used. Dedicated hardware solutions introduce additional implementation complexity that limits the efficiency of such vision algorithms. Processing at such a high frame rate is only applicable for tackling simple tasks such as the detection of the center of mass (or centroid) of moving objects [10], [11].

The newly developed event-based silicon retinas (DVS [7] and ATIS [5]) inspired by the physiology of biological retinas are promising sensors for fast vision applications. These sensors convey a sparse stream of asynchronous time-stamped events suitable for object tracking as only dynamic information is captured. Several tracking algorithms have been developed for this type of sensor. An event clustering algorithm is introduced for traffic monitoring, where clusters can change in size but are restricted to a circular form [12], [13]. A fast sensory motor system has been built to demonstrate the sensor's high-temporal resolution properties in [14]. Several event-based algorithms and a remarkable JAVA framework for the DVS can be found at [15]. In [16], a pencil balancing robot is developed to stabilize a pencil using a fast event-based Hough transform. The sensor has been recently applied to track particles in microrobotics [17] and fluid mechanics [18]. It was also used to track a microgripper's jaws to provide a real-time haptic feedback from the micrometer scale world (lengths and sizes of objects $\sim 1e$-6 m) [19].

To date, the currently developed methods that operate on events focus on the extraction of features such as lines in the whole focal plane. In other cases, they are too general to deal with particular cases, where trackers should locally follow a specific oriented edge or local shape. We will then extend the existing work to provide a more general framework allowing the tracking of specific local features using an event-based methodology. The tracking approach proposed here is inspired by the mean-shift technique that has also been extensively used in conventional frame-based visual tracking [20]–[22].

B. Gaussian Blob Tracking

A stream of visual events can be mathematically defined as follows: let $\text{ev}(\mathbf{u}, t) = [\mathbf{u}, t, \text{pol}]^T$ be a quadruplet giving the pixel position $\mathbf{u} = [x, y]^T$, the time t of the event, and pol, its polarity that can be -1 or 1. When an object moves, the pixels generate events, which geometrically form a point cloud that represents the spatial distribution of the observed shape. A moving object generates events that follow a spatial distribution that can be, in a first stage, roughly approximated by a bivariate normal distribution $\mathcal{N}(\mu, \Sigma)$, also called bivariate Gaussian distribution. The parameters of $\mathcal{N}(\mu, \Sigma)$ provide the object's position, size, and orientation. The Gaussian mean μ indicates the object's position, whereas the covariance matrix Σ represents its size and orientation. Let

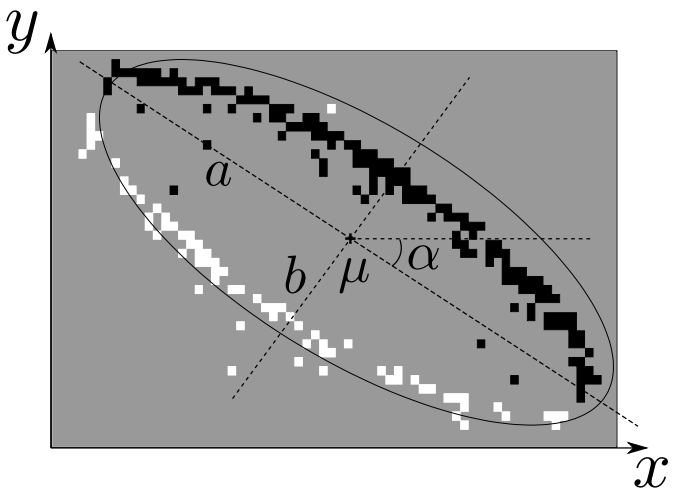


Fig. 2. Error ellipse of the covariance matrix of a Gaussian tracker following a black ellipse moving under a white background. Position, size, and orientation of the tracker automatically adapt to the visual stimuli. Black and white dots represent, respectively, OFF and ON events.

us suppose that several Gaussian trackers have already been initialized on the focal plane's locations of several moving objects. When a new event occurs, it is assigned a score (up to the normalization term) for each tracker, inspired by the probability that this event would be generated by the Gaussian distribution associated to this tracker, which can be calculated by

$$p_i(\mathbf{u}) = \frac{1}{2\pi}|\Sigma_i|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{u}-\mu_i)^T \Sigma^{-1}(\mathbf{u}-\mu_i)} \tag{1}$$

where $\mathbf{u} = [x, y]^T$ is the pixel location of the event. $\mu_i = [\mu_{ix}, \mu_{iy}]^T$ represents the ith tracker's location and $\Sigma_i \in \mathbb{R}^{2\times 2}$ is its covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix}. \tag{2}$$

Fig. 2 is showing an example of a ellipse shape tracked by such a Gaussian tracker.

If one of the computed scores is superior to a predefined threshold $p_i(\mathbf{u}) > \delta p$ (usually set to 0.1), then this tracker will adapt its parameters to follow this incoming event. If several trackers respond to the same event, the one with the maximum score is always chosen. After the most probable tracker has been selected, its parameters can be updated using a simple weighting strategy by integrating the last distribution with the current event information [see (3) and (4)]. Since only the chosen tracker is examined hereafter, the subscript i indicating the tracker number is omitted for clarity. The position and size of a Gaussian tracker can be updated as follows:

$$\mu_t = \alpha_1 \mu_{t-1} + (1-\alpha_1)\mathbf{u} \tag{3}$$

$$\Sigma_t = \alpha_2 \Sigma_{t-1} + (1-\alpha_2)\Delta\Sigma \tag{4}$$

where α_1 and α_2 are the update factors. They should be adjusted according to the event rate and the nature of the observed scene. These values are typically set between 10^{-2} and 10^{-1} and are chosen according to the size and velocity of the tracked objects. Namely, how many events should be received to drag the shape from its old position to the new one and to change its size and shape according to the incoming events' rates.

The covariance difference $\Delta\Sigma$ is computed using the current tracker's location $\mu_\mathbf{t} = [\mu_{tx}, \mu_{ty}]^T$ and event's location $\mathbf{u}$

$$\Delta\Sigma = \begin{bmatrix} (x-\mu_{tx})^2 & (x-\mu_{tx})(y-\mu_{ty}) \\ (x-\mu_{tx})(y-\mu_{ty}) & (y-\mu_{ty})^2 \end{bmatrix}. \tag{5}$$

Finally, we define the activity of each tracker $\mathcal{A}_i$ that is updated at each incoming event $\text{ev}(\mathbf{u}, t)$, following an exponential decay function, which describes the temporal dimension of the Gaussian kernel:

$$\mathcal{A}_i(t) = \begin{cases} \mathcal{A}_i(t-\Delta t)e^{-\frac{\Delta t}{\tau}} + p_i(\mathbf{u}), \\ \quad \text{if ev}(\mathbf{u}, t) \text{ belongs to tracker } i \\ \mathcal{A}_i(t-\Delta t)e^{-\frac{\Delta t}{\tau}}, \quad \text{otherwise} \end{cases} \tag{6}$$

where Δt is the time difference between current and previous events and τ tunes the temporal activity decrease. This activity measure is useful for inhibition and repulsion procedures that will be explained latter. It allows us to shape the interaction between trackers.

Remark: The object's size and orientation can be retrieved by computing the principal components of the covariance matrix. Computationally, the lengths of the error ellipse's axes are the two principal components of the covariance matrix, which can be explicitly calculated by decomposing the two eigenvalues $\lambda_{\min}$ and $\lambda_{\max}$. The semiaxes a and b and the orientation angle α of the ellipse can be computed as follows:

$$a = K\sqrt{\lambda_{\max}} \tag{7}$$

$$b = K\sqrt{\lambda_{\min}} \tag{8}$$

$$\alpha = \frac{1}{2}\arctan\left(\frac{2\sigma_{xy}}{\sigma_y^2 - \sigma_x^2}\right) \tag{9}$$

where K is a scaling factor describing the distribution's confidence level. K is given by the confidence intervals and it provides a tuning parameter linking the Gaussian distribution parameters to the size of the event cloud it is fitted to in the image plane. In practice, this value can be set to 1. Readers wishing to know more about the computation of confidence intervals should refer to [23].

Computing these parameters can provide an additional information if the trackers are used to identify unknown objects in a scene. The size and orientation of the Gaussian distributions yield information about the shape or orientation of the tracked objects and can be used either to discriminate between interesting and irrelevant objects or to build more complex objects from several trackers.

C. Multikernel Features Tracking

1) Principle: The principle is very similar to the Gaussian tracker, except that the Gaussian kernel is replaced by various other kernels. The examples that will be illustrated here are Gabor-oriented kernels, combinations of Gabor functions, Laplacian of Gaussian (LoG), and more general handmade kernels such as a triangle and square. Some of these kernels are shown in Fig. 3.

The feature tracker generalizes to almost any kernel even for kernels with no analytical form. If K_i has no closed form, then

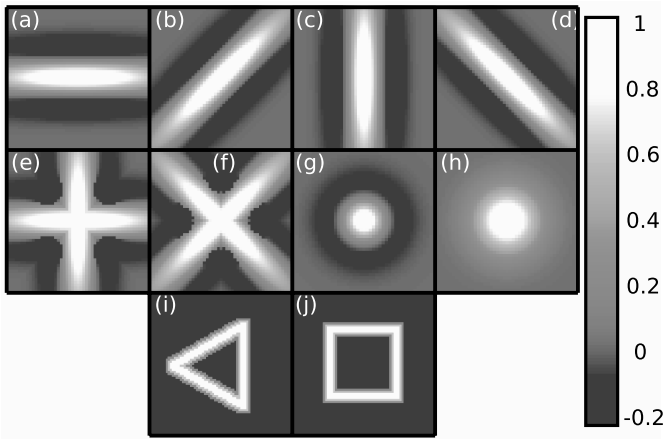


Fig. 3. Examples of kernels used for tracking. (a)–(d) Gabor-oriented kernels. (e) and (f) Combinations of Gabor kernels. (g) LoG. (h) Gaussian kernel. (i) and (j) Handmade arbitrary kernels.

it is a numerically defined function. In this case, the update of size and orientation cannot be performed easily as this implies the estimation of rotation and scaling that go beyond the scope of this paper. We chose, in this case, to restrict kernels to a fixed size (scale) and orientation. Due to the low computational cost, this can easily be supplemented by the use of multiple layers of kernels of different scales and orientations as will be shown in Section IV.

We introduce a local inhibition procedure that prevents the tracker from being active if the same localized area of the kernel is always excited. The activity of tracker i is then given by

$$\mathcal{A}_i(t) = \begin{cases} \mathcal{A}_i(t-\Delta t)e^{-\frac{\Delta t}{\tau}} + G_{\text{inhib},i}(\text{ev},\mathbf{u})K_i(x_i - x, y_i - y), \\ \quad \text{if } ev(\mathbf{u},t) \text{ occurs into receptive field of tracker } i \\ \mathcal{A}_i(t-\Delta t)e^{-\frac{\Delta t}{\tau}}, \quad \text{otherwise} \end{cases} \tag{10}$$

where $\mathbf{u} = [x_i, y_i]^T$ is the position of tracker i and K_i is the matrix representing the tracker's kernel and G_{inhib}, the inhibition gain, is computed as follows:

$$G_{\text{inhib},i}(ev,\mathbf{u}) = 1 - e^{-\Delta t_{ev,\mathbf{u}}/\tau_{\text{inhib}}}. \tag{11}$$

Local inhibition is ensured considering $\Delta t_{\text{ev},\mathbf{u}}$ as the temporal difference between event ev and last occurrence of an event into a small spatial neighborhood of the point $[x_i - x, y_i - y]^T$ in matrix K_i (we usually considered a neighborhood of one or two pixels radius). This mechanism is equivalent to a local inhibition of neighboring locations of matrix K_i, where the event occurred. Namely, the activated location of the filters and its surrounding will no longer be able to drive the tracker even if an event reactivates that location. In that case, $\Delta t_{\text{ev},\mathbf{u}}$ will go toward 0 thus leading G_{inhib} to 0. As the temporal distance increases, G_{inhib} will gradually tend toward 1. This ensures that a kernel needs to receive events in all its area to lead to the tracker's activation. This inhibition is followed by an exponential reactivation of the inhibited area thus allowing the filter to be active to events arriving at that location.

2) Gabor-Oriented Kernels and Combinations of Gabors: It is well known that some areas of our visual cortex V1 respond preferentially to oriented stimuli [24]. Consequently, we used Gabor functions to build a model of orientation selective kernels. The response of a θ-oriented Gabor kernel $\mathcal{K}_{G\theta}(\mathbf{v},\sigma)$ located at position $\mathbf{v} = [x_G, y_G]^T$ to the incoming event $\text{ev}(\mathbf{u},t)$ is given by

$$G(e,\mathcal{K}_{G\theta}(\mathbf{v},\sigma)) = e^{\left(-\frac{x_{\mathbf{u},\mathbf{v}}^2 + \gamma^2 y_{\mathbf{u},\mathbf{v}}^2}{2\sigma^2}\right)} \cos\left(2\pi \frac{x_{\mathbf{u},\mathbf{v}}}{\lambda}\right) \tag{12}$$

where $x_{\mathbf{u},\mathbf{v}} = (x - x_G)\cos\theta + (y - y_G)\sin\theta$ and $y_{\mathbf{u},\mathbf{v}} = -(x - x_G)\sin\theta + (y - y_G)\cos\theta$.

To ensure a correct orientation selectivity, we set the parameters $\gamma = \sigma/15$ and $\lambda = 4\sigma$. Gabor kernels are illustrated in Fig. 3(a)–(d). To track a particular feature like a cross, we also built kernels by combining Gabor functions following orthogonal orientations, as shown in Fig. 3(e) and (f).

3) LoG Kernel: This kernel is inspired by center-surround biological structures and can be represented by the LoG function, shown in Fig. 3(g). This kernel has a behavior similar to Gaussian kernels shown in Fig. 3(h) but is also sensitive to the size of the tracked events' cloud. A cloud whose size exceeds the bright central ring [Fig. 3(g)] will induce negative contributions to the activity of events occurring in the dark ring area. This compensates for the effect of centered events, preventing the tracker from being activated.

4) General Kernels: In fact, the algorithm can use any kernel, with the only constraint being that the center of mass of the matrix that represents the kernel that has to be aligned with the center of the kernel. This is a weak constraint since any general kernel can be spatially shifted to meet this restriction arising from the position's update principle. Since the tracker is attracted by the neighboring events, its position will naturally match the local events' center of mass. To test if the events' cloud matches the desired feature, both must share the same spatial locations.

As an example, general kernels are shown using a square and triangle [Fig. 3(i) and (j)]. These kernels are generated using a binary representation of a simple geometric shape (1 on lines and -1 elsewhere) followed by a dilatation algorithm and smoothing.

D. Multitarget Tracking

The algorithm is first initialized with a hidden layer of preconstructed trackers that are uniformly distributed among the whole field-of-view. Hidden trackers refer to those not displayed on the screen as active trackers. A hidden tracker is one that does not represent a real object but serves to seed a potential tracker.

A tracker will be automatically attracted to the nearest events' cloud. When a new event occurs, and no active tracker responds to it, it directly feeds the nearest hidden tracker with the same update rule as described by (3) [Fig. 4(b)]. When the activity of a hidden tracker increases above a predefined threshold $\mathcal{A}_{\text{up}}$ [Fig. 4(c)], the tracker is upgraded to the visible layer, and a new hidden tracker is added with default parameters so that the hidden layer always has a fixed number

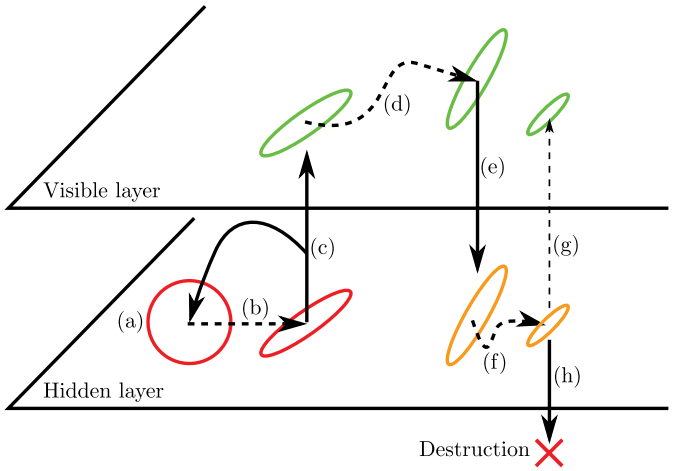


Fig. 4. Activation scheme of trackers (Gaussian tracker in the example). (a) Tracker is initialized in the hidden layer with default parameters. (b) It starts to follow the event cloud from a moving object and adapts to its shape. (c) When its activity goes above $\mathcal{A}_{\text{up}}$, the tracker is upgraded to the visible layer and a new independent tracker is created in the hidden layer with its initial position and parameters. (d) Tracker follows the event cloud. (e) When its activity decreases and falls under $\mathcal{A}_{\text{up}}$, the tracker is downgraded to the hidden layer but keeps its current parameters and position. (f) Another step of tracking. (g) Eventually, the activity may rise again over $\mathcal{A}_{\text{up}}$ and the tracker can be upgraded again. (h) If its activity falls under $\mathcal{A}_{\text{down}}$, the tracker is deleted.

Algorithm 1 Event-Based Features Tracking Algorithm

```
for every incoming event ev(u, t) do
  Update the activity A_i of each tracker of the visible layer
  using Eq (6) or (10).
  Update the best candidate tracker's position (and size)
  using equations (3) (and (4)).
  for every tracker of both layer do
    if A_i > A_up then
      Upgrade tracker to visible layer.
    else if A_i < A_down then
      Delete tracker i
    else
      Downgrade tracker to hidden layer, and keep the same
      parameters and position.
    end if
  end for
  for every pair of trackers of both layers do
    Update tracker position using repulsion equation (13)
  end for
  for every tracker of hidden layer do
    Update tracker position using attraction equation (14)
  end for
end for
```

of trackers distributed across the scene. When its activity falls under $\mathcal{A}_{\text{down}}$, the visible tracker is deleted [Fig. 4(h)]. In the intermediary step, between $\mathcal{A}_{\text{up}}$ and $\mathcal{A}_{\text{down}}$ [Fig. 4(f)], the tracker is downgraded to the hidden layer but keeps its actual parameters and position. The evolution criteria combine both spatial and temporal correlation of events. Parameters $\mathcal{A}_{\text{down}}$, $\mathcal{A}_{\text{up}}$ and the temporal decrease constant τ have to be tuned according to the event rate and desired behavior.

E. Mutual Repulsion and Attraction to Initial Position

In case of strong localized activity, all neighboring trackers can be attracted to the same location. In order to prevent the different trackers following the same cloud, a mutual repulsion process is added. Each time a tracker is activated by an event, the distance between each tracker pair is computed and the trackers that are too close are shifted away from each other, following the weighted repulsion function described in (13). The location μ_i of a tracker is updated compared with the location of another tracker at μ_j as follows:

$$\mu_{\mathbf{i}} \leftarrow \mu_{\mathbf{i}} - \alpha_{\text{rep}} e^{-\frac{\|\mu_{\mathbf{i}} - \mu_{\mathbf{j}}\|}{d_{\text{rep}}}} \frac{\mathcal{A}_j^2}{\mathcal{A}_i^2 + \mathcal{A}_j^2} (\mu_{\mathbf{j}} - \mu_{\mathbf{i}}) \tag{13}$$

where parameters α_{rep} and d_{rep} set the repulsion behavior. The mutual repulsion between trackers i and j is designed to ensure a correct repartition of trackers based on the distance that separates them, but also on the activity of both trackers. The more a tracker is active (i.e., is efficiently tracking an event cloud), the less it will be influenced by its neighbors.

Due to this mutual repulsion, inactive trackers are often pushed far from their initial position by moving active trackers. Thus, empty areas could grow on the field-of-view. To compensate this possibility, we also added an attraction force that impacts on each inactive tracker in the following way:

$$\mu_{\mathbf{i}} \leftarrow \begin{cases} \mu_{\mathbf{i}}^{\mathbf{0}}, & \text{if } \|\mu_{\mathbf{i}} - \mu_{\mathbf{i}}^{\mathbf{0}}\| > d_{\max} \\ \mu_{\mathbf{i}} + \alpha_{\text{att}}(\mu_{\mathbf{i}}^{\mathbf{0}} - \mu_{\mathbf{i}}), & \text{otherwise} \end{cases} \tag{14}$$

where $\mu_{\mathbf{i}}^{\mathbf{0}}$ is the initial position of tracker i. Thus, an inactive tracker that moved too far away from its initial position without being activated will be reset to its initial position.

Two additional constraints are added. Trackers that are near the border and about to move out of the focal plane are deleted. The borders are one pixel thick and a tracker is deleted if the distance of the its center to the border is below its typical size. For a Gaussian blob, this typical size is the length of the ellipse's smallest axis (axis b in Fig. 2). For a symetric tracker (e.g., square and triangle), the size is the circumscribed circle's radius. The second constraint prevents the Σ matrix going below a low threshold value for the Gaussian trackers to limit the tracking to objects with a reasonable size. These additional constraints do not have much impact on the overall tracking performance as the sensor is usually configured to capture the scene in center of the focal plane.

F. Global Algorithm

To summarize, the whole process of event-based features tracking is given by Algorithm 1.

1) Remarks: For simplicity, repulsion and attraction are computed every time a new event occurs, but as trackers do not move that fast, it is possible to perform computation less frequently. When adding this optimization, processing the repulsion and attraction step can usually be done every millisecond and still yield good performances.

Events are considered regardless of their polarity as we are interested in the global position and orientation of the object.

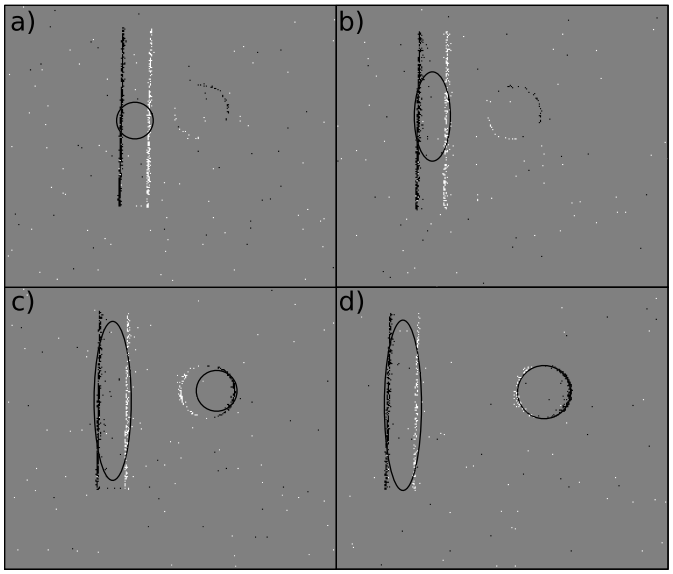


Fig. 5. Events cloud accumulated over 15 ms (for each subfigure), ON and OFF events are represented by white and black points, respectively. Gaussian trackers are attracted and deformed in response to the distribution of events clouds corresponding to different objects. (a) Activity of a hidden tracker crosses the $\mathcal{A}_{\text{up}}$ threshold, it then becomes visible. (b) Tracker rapidly adapts to the events cloud's distribution. (c) and (d) While the first tracker matches the rectangle object, another object starts to move, causing a second tracker to update.

In what follows, we will focus on a single polarity. This will be carried out to provide a precise measure of accuracy of the positioning of a tracker for a thin contour.

IV. Results

The experiments have been carried out using the ATIS camera considering only its change detection output and gray levels have not been used. All programs have been written in C++ under Linux. We experimentally set the parameters related to the activity of trackers to $A_{\text{up}} = 0.15$ and $A_{\text{down}} = 0.1$. Parameters related to the extraction of orientations from the covariance matrix have been set to $K = 1$ [see (7), (8)] and $\tau = 20$ ms (6). The repulsion and attraction rates have been set to $d_{\max} = 40$ (14) and $d_{\text{rep}} = 20$ (13). These have been set according to the mean size of observed objects. The repulsion and attraction rates have been set to $\alpha_{\text{att}} = 0.01$ (14) and $\alpha_{\text{rep}} = 0.1$ (13). Parameters $\tau = 5$ ms (10) and $\tau_{\text{inhib}} = 10$ μs (11) have low values so that trackers adapt to fast motions.

As in every low-level image processing technique, these values can unfortunately only be experimentally adjusted from the statistics of observed scenes. We found these values to be optimal for all performed experiments. They appear to comply with a large variety of conventional indoor environments for fast and slow stimuli.[1]

A. Gaussian Blob Trackers

Fig. 5 shows the Gaussian trackers growth and adaptation to events clouds for two independently moving objects (a rectangle and disc; note that the rectangle only produces two lines of events because it moves in a parallel direction to its two missing edges). The settings of parameters were adjusted experimentally to: 1) $\alpha_1 = 0.2$ (3) and 2) $\alpha_2 = 0.01$ (4). The adjustment factors have been set smaller for the covariance matrix so that the trackers' size does not collapse in case few events are generated by the target. In practice, the value of the position of the tracker has to move fast to keep up with the motion, while we assume that the shape of the object will change at a slower rate.

[1]This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. This consists of a video showing the described tracking methods operating in different conditions such as real-world data and some of the experiments presented in this paper. This material is 12 MB in size.

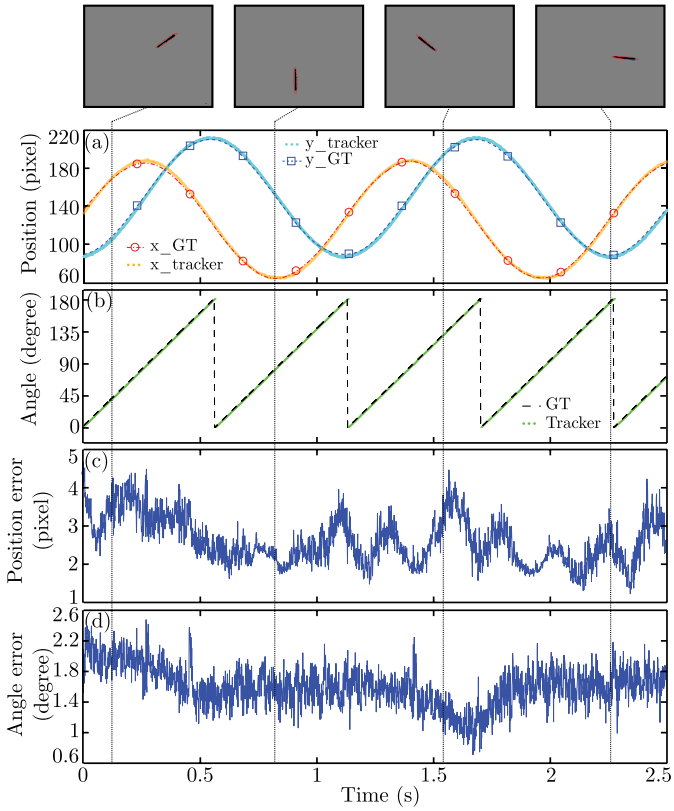


Fig. 6. Gaussian tracker accuracy in position and angle. (a) Object position in x (dashed line with circle markers) and y (dashed line with square markers). Underlying dots represent the tracker's positions. They are hardly distinguishable from the GT measure. (b) Object's angle (dashed line) and tracker's angle (underlying dots). (c) and (d) Tracking error, respectively, in position and angle computed every millisecond. On top are shown snapshots (accumulation of events during 10 ms) of the segment at different timestamps during motion and the ellipse representing the active Gaussian tracker's position, size, and orientation.

We evaluated the Gaussian tracker accuracy, for position, angle, and size. In a first experiment, the object tracked is a segment rotating along a circle at constant speed. During 10 rotations, only one Gaussian tracker activated and followed the segment. As shown in Fig. 6, ground truth (GT) position in x and y (respectively, lines with circle and square markers) are barely distinguishable from the tracker's position, which is output every millisecond for display purpose but internally updated asynchronously every time an event occurred. The position tracking error is computed as the Euclidian distance between the segment's GT position and tracker's position every millisecond: (2.61 ± 0.62 pixels, mean $\pm$ STD). The segment angle [dashed line in Fig. 6(b)] is also tracked (underlying points) with a very high precision ($1.59° \pm 0.26°$). Due to

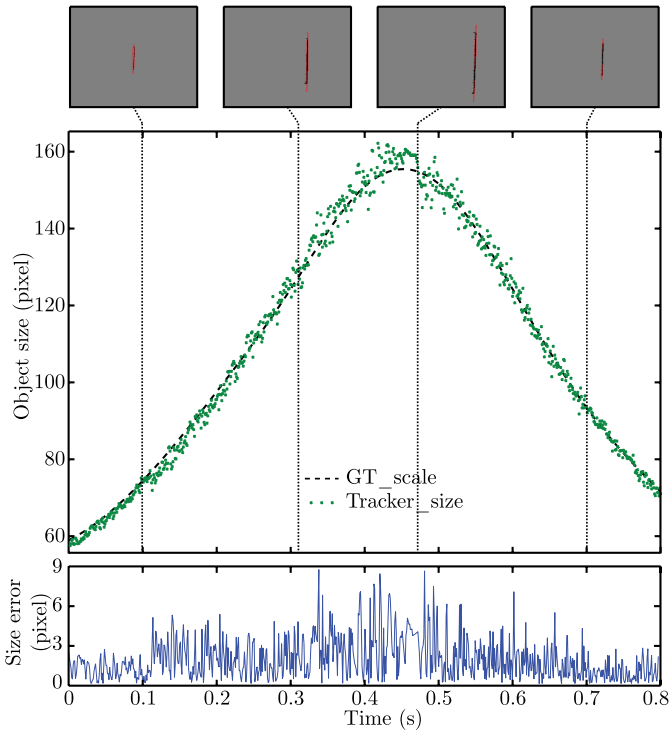


Fig. 7. Gaussian tracker accuracy in size. (a) Object's GT size (black dashed line), and Gaussian tracker's major axis (underlying dots). (b) Size's error computed every millisecond. On top are shown snapshots (accumulation of events during 10 ms) of the segment at different timestamps during motion. Ellipse: Gaussian tracker.

symmetries, angles are shown from 0° to 180°. GTs position, angle, and scale were measured by hand clicking the segment's extremities periodically on reconstructed frames.

To measure the tracking error using the Gaussian tracker, we moved the same line segment closer and further in front of the retina so that its length changes as we moved closer and farther away. Fig. 7 shows the GT evolution of the length of the line segment (black dashed line) and the corresponding Gaussian tracker's major axis (small dots). The corresponding tracking error (bottom curve) is 1.9 ± 1.6 pixel.

Finally, we performed an experiment where a pen is thrown in front of the camera. The result of its real-time tracking is shown in Fig. 8. The pen is successfully tracked in real time. Some trackers are activated by the person's motion such as his head and hand that are then also tracked. The mean error of tracking is 1.2 pixel, it corresponds to the distance between the gravity center of the pen and that of the tracker. The GT has been performed by manually labeling the gravity center of the pen on reconstructed frames.

B. Gabor Kernels

All Gabor trackers sizes have been initially set to $\sigma = 3$ with $\gamma = \sigma/15$ and $\lambda = 4\sigma$ (12). The kernel for each orientation has a size of 17×12 pixel and is constant through the experiment (so the size and orientation of each filter will not change during tracking). Four Gabor-oriented kernels (Fig. 9) are used to track the thrown pen. As shown in Fig. 9, the oriented trackers successfully track the rotating pen. From a handmade GT, we found that the mean distance between the position of tracked-oriented edges and center of trackers has a mean value of 1.4 pixel. The oriented filters show a 100% of success in detecting orientation.

C. Gabor Combinations and General Kernels

In order to evaluate the spatiotemporal precision of the algorithm, we tested it with more complex kernels. We used combinations of Gabor functions [crosses in Fig. 3(e) and (f)] and more general handmade kernels [triangle and square in Fig. 3(i) and (j)] were used with a 6×3 initialization grid of hidden trackers for each type of kernel. As in previous experiments, active trackers are represented in Fig. 10 by corresponding superimposed shapes. The stimuli were printed on a sheet of paper that was held by hand and moved in front of the sensor. During the experiment, one unique tracker per feature was active, it successfully followed its corresponding shape.

Fig. 11 shows the spatiotemporal precision of the algorithm, showing the GT (line with circle markers) and corresponding active tracker's positions over time (small dots). The GT is computed every 10 ms (line with circle markers) accumulating OFF events between two measures and hand clicking the feature center. We used a single polarity to provide a precise measurement of the accuracy of the trackers when following a single contour. In order not to overload Fig. 11, the tracker's position is showed every 1 ms. The activities and positions are updated asynchronously every time an event occurs.

The tracking error is computed as the Euclidian distance between the tracker's position and GT. It is shown in Fig. 12 for all used kernels. The two cross kernels show a low error of 0.81 ± 0.42 pixel and 0.70 ± 0.38 pixel (mean $\pm$ STD). The error of the square kernel is a bit higher (1.12 ± 0.66 pixel) but remains lower than one pixel. However, the last kernel (triangle) has a significantly higher tracking error. This is again due to the event-based acquisition mechanism. When the relative motion is parallel to a segment of the shape, no event is acquired from this segment. As shown in Fig. 13, during vertical motion, the vertical segments of the triangle, rectangle, and straight cross disappear.

Fig. 13 shows the motion of the object (dashed circle) and the different positions (segments along the circle). These positions are also reported on Fig. 12 and correspond to the configurations that lead to an increase of the triangle tracking error (bottom plot).

D. Computational Cost

We estimated the average time per event spent to update each tracker by measuring the total computation time required by the algorithm in experiments showing a uniform event rate when using a single computation thread. On the computer used in the experiments (Intel Xeon E5520 at 2.27 GHz), this time is ~25 ns/event and per tracker for a kernel size of 70×70 if the kernel's matrix has been precomputed and processing regulation steps (trackers' repulsion/attraction every millisecond) when running on a single core. Consequently, in a typical natural scene (that statistically generates 200 000 events/s), we estimate that it is possible to compute in real time around 200

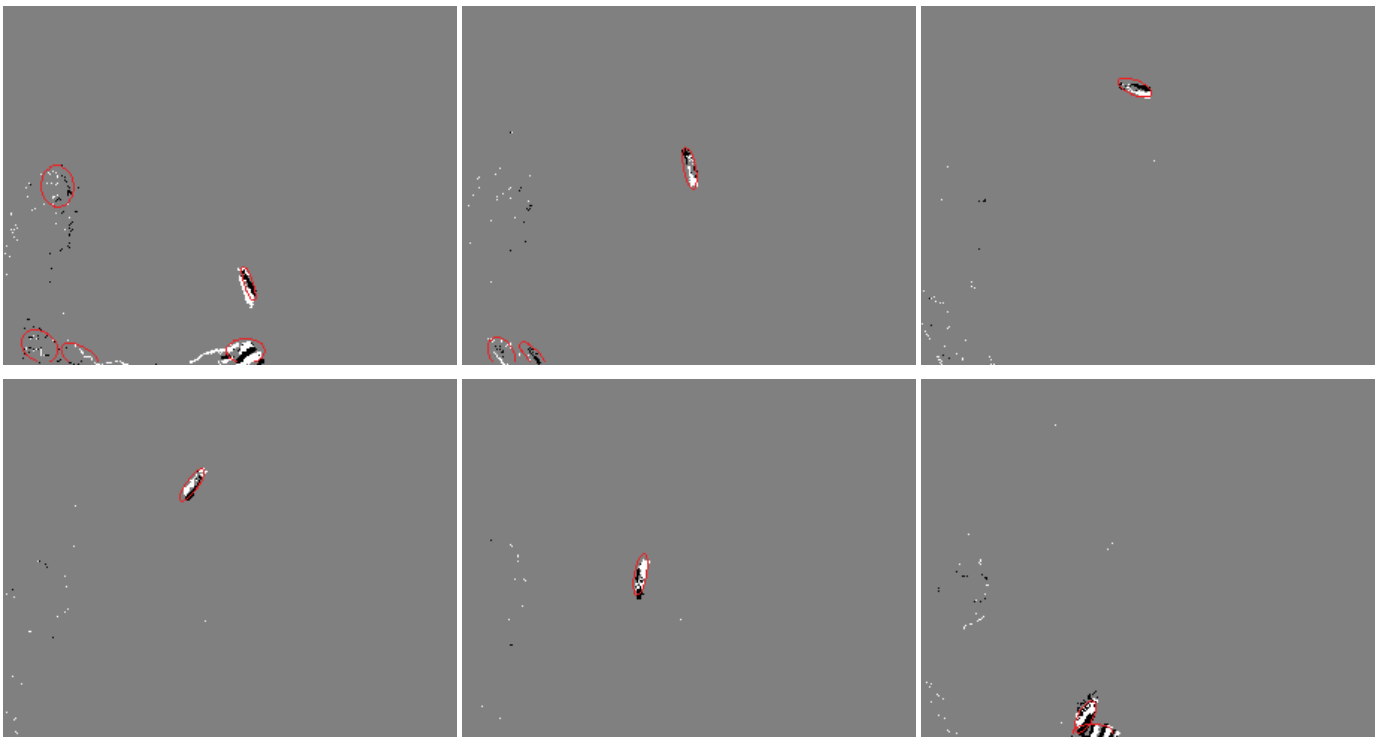


Fig. 8. From top to down and left to right: sequence of snapshots, where the Gaussian blob trackers follow a pen thrown in the air. The trackers stick to the pen's position. Blobs are also activated when the person on the left of the scene is moving. They are successfully tracking the person's features and eventually disappear in absence of motion.

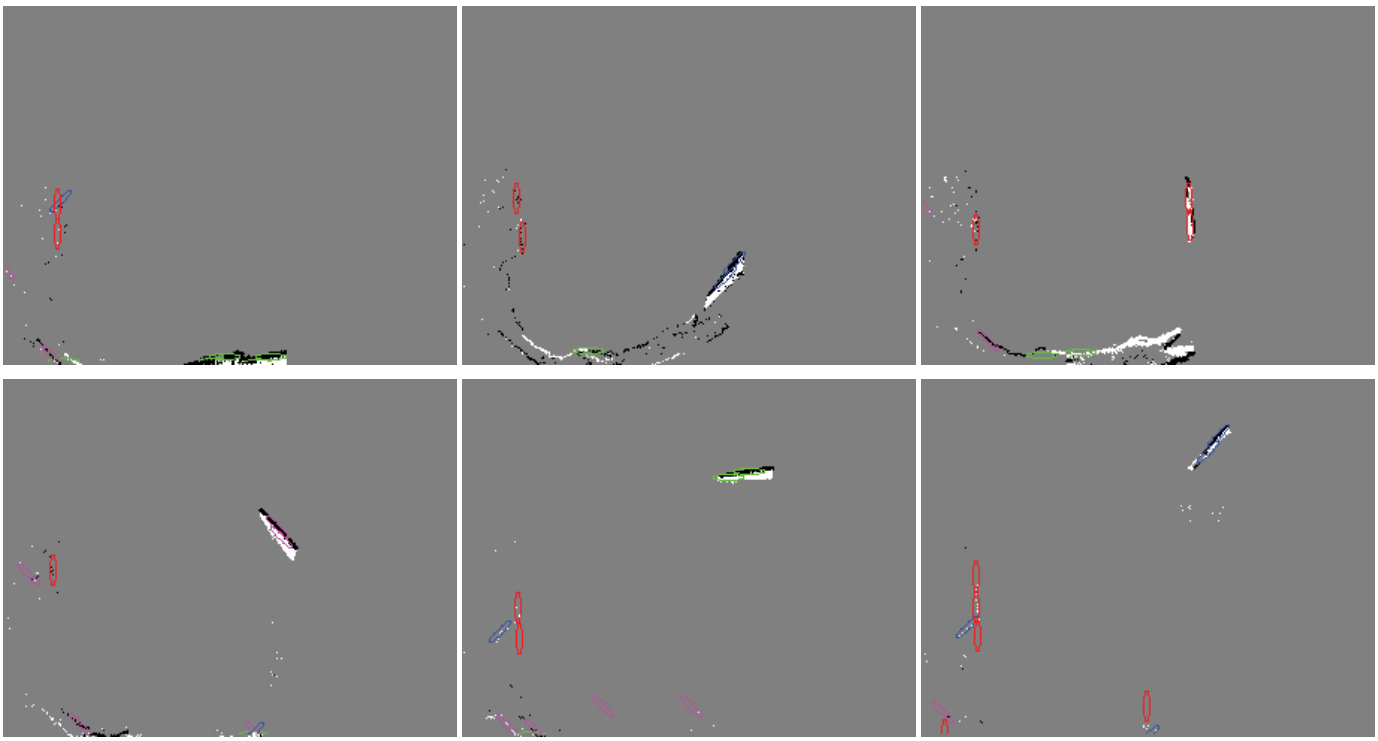


Fig. 9. From top to down and left to right: sequence of snapshots where oriented Gabor trackers follow the rotation of a pen thrown in the air. The orientation of the pen is successfully tracked. This experiment uses four pools each containing one particular Gabor tracker with a given size and orientation (though tracking a total of four different orientations independently). Oriented trackers are activated sequentially while the pen performs a full 360° rotation. The orientation filters are also activated by the person's movements when they happen. Note, how the kernels track the pen but never overlap. This is due to the repulsion mechanism.

of these kernels with one single core of our CPU. The total performance of a multithreaded implementation using all of the CPUs cores has not been verified experimentally.

The Gaussian trackers are obviously the most computationally expensive ones due to the need to provide a value of the exponential. For the other filters, the computation

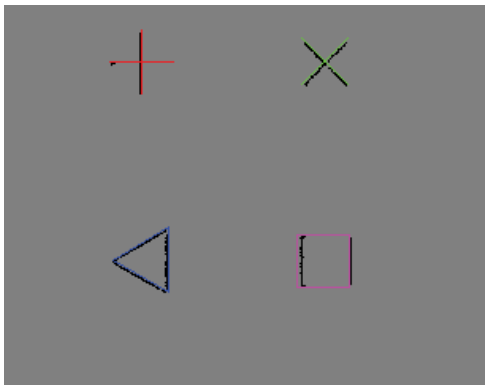


Fig. 10. Tracking results for specific features using combinations of Gabors and general kernels. Each active kernel is represented by the corresponding shape. The snapshot has been created accumulating events (black dots) during 10 ms.

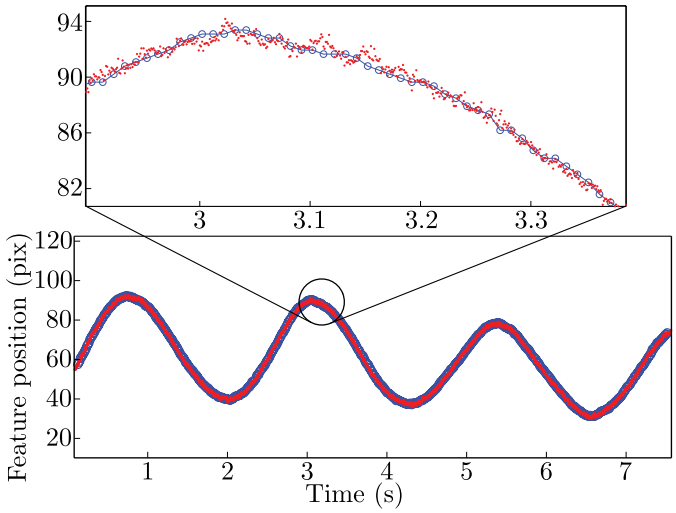


Fig. 11. Horizontal position of the first active tracker (straight cross in top left corner of Fig. 10) over time (small dots), and the position of the tracked shape (line with circle markers).

costs are similar. We precomputed the kernel matrices changes for every position of the incoming event in the kernel. This turns out to be computationally inexpensive as the cost becomes that of accessing a lookup table in memory.

E. Retrieving Feature Scale and Orientation

Feature trackers use kernels with fixed size and orientation, but it is also possible to track a feature even if its size and orientation evolve during time by computing in parallel multiple layers of trackers. Each layer behaves as described in previous sections. This allows to track features' size and orientation in a discretized space. In Fig. 14, we show the continuous tracking of a cross that rotates around its center at a constant speed. Due to the cross radial symmetry, we discretized the orientation space into 10 orientations from 0° to 180° and also added the GT.

The orientation of active tracker regularly alternates to follow the feature's orientation. We can in principle merge the 10 oriented trackers used here into a single rotation-invariant

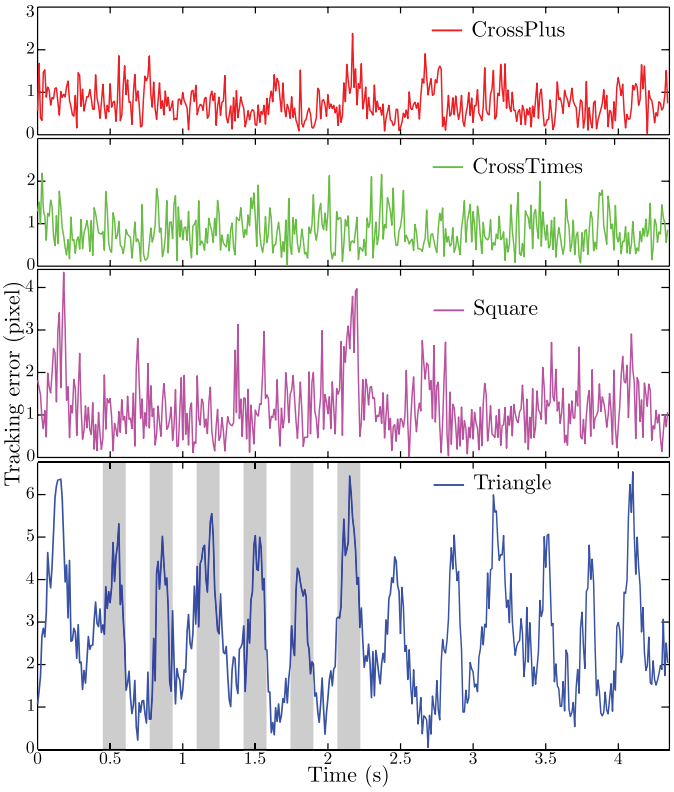


Fig. 12. Tracking errors for all used features. The error is computed as the Euclidian distance between the tracker's position and GT position of the corresponding shape. Gray areas: periods of time of a single rotation. When the motion direction is parallel to one of the triangle's segments, this leads to an increase of the tracking error. This is an expected result.

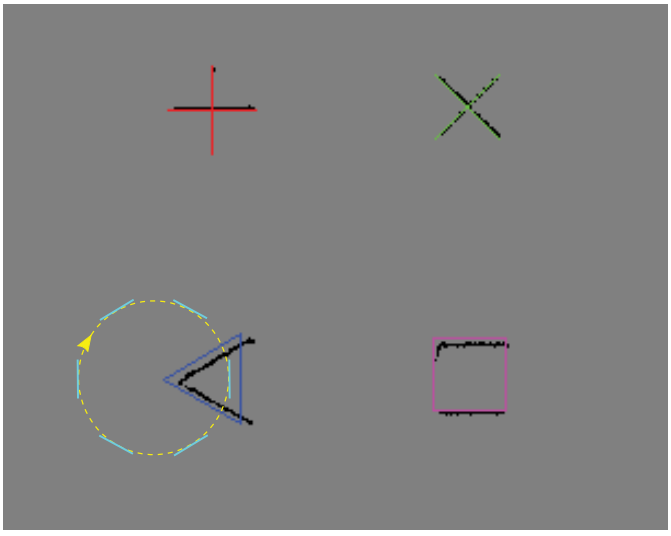


Fig. 13. Active trackers' positions (straight shapes) and corresponding GT (black dots). Dashed circle: approximate triangle's motion. Segments on this circle: positions where motion is parallel to one side of the triangle, leading to an increase of tracking error.

tracker for this cross by computing a global activity as the maximum of each oriented tracker activity.

Similarly, we can create scale-invariant feature trackers by combining multiple trackers whose kernels represent a unique feature at different scales. Fig. 15 shows the continuous

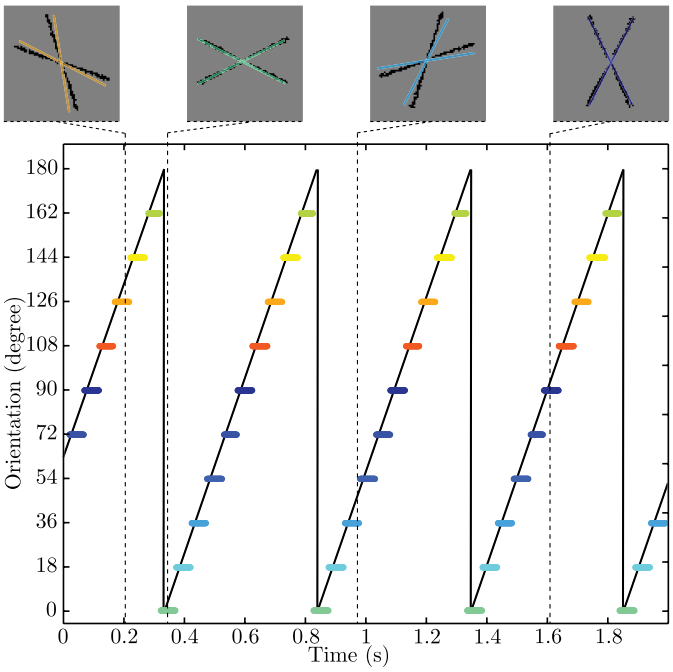


Fig. 14. Continuous tracking of a rotating cross. Each point on the bottom plot represents the activation of an oriented tracker. Its orientation is represented by its level. The orientation's GT (black line) was measured every 100 ms and interpolated considering constant rotational speed. Top images represent accumulations of OFF events during 10 ms at different timestamps and corresponding active trackers. Only OFF events are used here to ensure a precise measurement of trackers to a single contour.

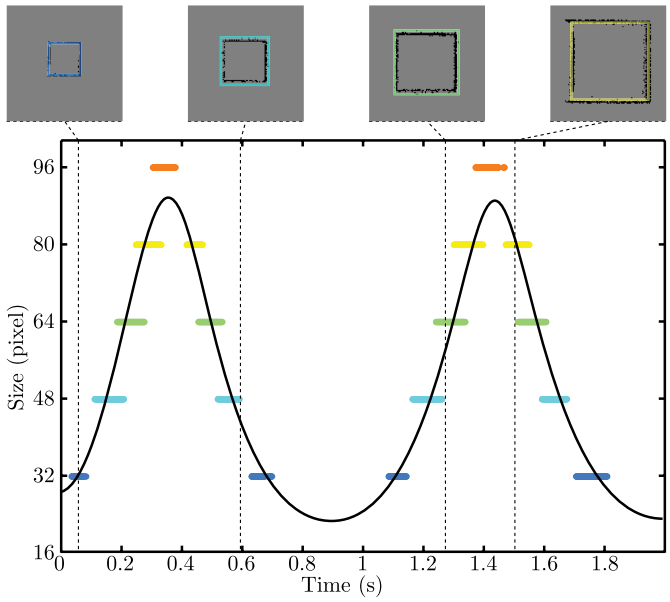


Fig. 15. Continuous tracking of a moving square. Each point on the bottom plot represents the activation of a specific scale tracker. Its size is represented by its level. The GT (black line) was measured every 10 ms. Top images represent accumulations of events during 10 ms.

tracking of a square that was moved back and forth in front of the camera. Six scaled square trackers were used corresponding to sizes of 16, 32, 48, 64, 80, and 96 pixels.

V. Conclusion

This paper presents an event-based methodology to track shapes. It allows the following of specific shapes at a very low computational cost thus matching the high temporal resolution of event-based vision sensors. It can adapt to orientation and change of scale. Experiments have shown the stability and repeatability of the algorithm. Because of its low computational cost, the method can track multiple targets in parallel. The same technique is also used to make the algorithm scale and orientation invariant. The method is particularly adapted to applications such as robotics vision-based navigation, Simultaneous Localization And Mapping (SLAM), or object recognition. These usually impose tight constraints on computation times and energy consumption.

The method relies on several parameters that need to be tuned. This is generally the case for almost every image processing method. Free-parameter techniques in computer vision are still an open problem. In this paper, we provided a set of parameters that we have shown to be efficient for a wide variety of applications, including highly dynamic scenes.

This paper is currently being extended to link trackers together in a coherent spatial arrangement to match complex shapes such as faces and human body. These techniques are known as part-based approaches [25], they are currently too computationally greedy in the conventional frame-based approach to be implemented at several kilohertz. Finally, it is important to emphasize the particular innovation of the method. The architecture avoids the N^2 operations per event associated with conventional kernel-based convolutional operations with $N \times N$ kernels. Our transfer function-based approach enables us to compute (or retrieve from a lookup table) only one value per tracker for each incoming event, which is dependent on the position of the event with respect to the center of the trackers. This is a major feature as, due to the number of incoming events, a full convolution would prevent from real-time implementation on current off-the-shelf computers.

References

[1] U. Muehlmann, M. Ribo, P. Lang, and A. Pinz, "A new high speed CMOS camera for real-time tracking applications," in *Proc. IEEE Int. Conf. Robot. Autom.*, Barcelona, Spain, Apr./May 2004, pp. 5195–5200.

[2] H. Oku, N. Ogawa, M. Ishikawa, and K. Hashimoto, "Two-dimensional tracking of a motile micro-organism allowing high-resolution observation with various imaging techniques," *Rev. Sci. Instrum.*, vol. 76, no. 3, pp. 034301-1–034301-9, Mar. 2005.

[3] T. Delbruck, B. Linares-Barranco, E. Culurciello, and C. Posch, "Activity-driven, event-based vision sensors," in *Proc. IEEE Int. Symp. Circuits Syst.*, Paris, France, May/Jun. 2010, pp. 2426–2429.

[4] T. Delbruck, "Frame-free dynamic digital vision," in *Proc. Int. Symp. Secure-Life Electron., Adv. Electron. Qual. Life, Soc.*, Tokyo, Japan, Mar. 2008, pp. 21–26.

[5] C. Posch, D. Matolin, and R. Wohlgenannt, "A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, Jan. 2011.

[6] C. Posch, D. Matolin, and R. Wohlgenannt, "An asynchronous time-based image sensor," in *Proc. IEEE Int. Symp. Circuits Syst.*, Seattle, WA, USA, May 2008, pp. 2130–2133.

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

[8] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 5, pp. 416–434, May 2000.

[9] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison, "Real-time camera tracking: When is high frame-rate best?" in *Computer Vision—ECCV* (Lecture Notes in Computer Science), vol. 7578, A. W. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. New York, NY, USA: Springer-Verlag, 2012, pp. 222–235. [Online]. Available: http://dblp.uni-trier.de/db/conf/eccv/eccv2012-7.html

[10] T. Komuro, I. Ishii, M. Ishikawa, and A. Yoshida, "A digital vision chip specialized for high-speed target tracking," *IEEE Trans. Electron Devices*, vol. 50, no. 1, pp. 191–199, Jan. 2003.

[11] T. G. Constandinou and C. Toumazou, "A micropower centroiding vision processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 6, pp. 1430–1443, Jun. 2006.

[12] M. Litzenberger *et al.*, "Estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor," in *Proc. IEEE Intell. Transp. Syst. Conf.*, Toronto, ON, Canada, Sep. 2006, pp. 653–658.

[13] M. Litzenberger *et al.*, "Embedded vision system for real-time object tracking using an asynchronous transient vision sensor," in *Proc. 12th Signal Process. Edu. Workshop, 4th Digit. Signal Process. Workshop*, Teton National Park, WY, USA, Sep. 2006, pp. 173–178.

[14] T. Delbruck and P. Lichtsteiner, "Fast sensory motor control based on event-based hybrid neuromorphic-procedural system," in *Proc. IEEE Int. Symp. Circuits Syst.*, New Orleans, LA, USA, May 2007, pp. 845–848.

[15] *Open Source jAER Software Project*. [Online]. Available: http://jaerproject.net/, accessed Jan. 2013.

[16] 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 *Proc. IEEE Int. Symp. Circuits Syst.*, Taipei, Taiwan, May 2009, pp. 781–784.

[17] Z. Ni, C. Pacoret, R. Benosman, S. Ieng, and S. Régnier, "Asynchronous event-based high speed vision for microparticle tracking," *J. Microsc.*, vol. 245, no. 3, pp. 236–244, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1111/j.1365-2818.2011.03565.x

[18] D. Drazen, P. Lichtsteiner, P. Häfliger, T. Delbrück, and A. Jensen, "Toward real-time particle tracking using an event-based dynamic vision sensor," *Experim. Fluids*, vol. 51, no. 5, pp. 1465–1469, Nov. 2011. [Online]. Available: http://dx.doi.org/10.1007/s00348-011-1207-y

[19] Z. Ni, A. Bolopion, J. Agnus, R. Benosman, and S. Régnier, "Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics," *IEEE Trans. Robot.*, vol. 28, no. 5, pp. 1081–1089, Oct. 2012.

[20] M. Fashing and C. Tomasi, "Mean shift is a bound optimization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 3, pp. 471–474, Mar. 2005.

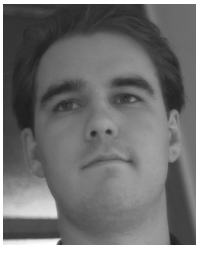
[21] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 5, pp. 564–577, May 2003.

[22] H. Zhou, Y. Yuan, and C. Shi, "Object tracking using sift features and mean shift," *Comput. Vis. Image Understand.*, vol. 113, no. 3, pp. 345–352, Mar. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.cviu.2008.08.006

[23] K. Krishnamoorthy, *Handbook of Statistical Distributions with Applications*. Boca Raton, FL, USA: Chapman & Hall, 2006.

[24] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.*, vol. 160, no. 1, pp. 106–154, Jan. 1962.

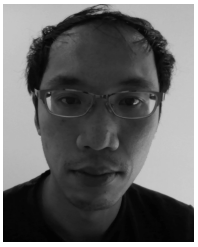
[25] M. A. Fischler and R. A. Elschlager, "The representation and matching of pictorial structures," *IEEE Trans. Comput.*, vol. C-22, no. 1, pp. 67–92, Jan. 1973. [Online]. Available: http://dx.doi.org/10.1109/T-C.1973.223602



Xavier Lagorce received the B.Sc. degree, the Agrégation degree in electrical engineering, and the M.Sc. degree in advanced systems and robotics from the École Normale Supérieure de Cachan, Cachan, France, in 2008, 2010, and 2011, respectively. He is currently pursuing the Ph.D. degree in neurally inspired hardware systems and sensors with the Vision Institute, Paris.

Cédric Meyer received the Ph.D. degree in computer vision from Pierre-and-Marie-Curie University, Paris, France, in 2013.

He was involved in event-based vision and its application to object recognition. The main idea was to try to improve vision sensing for embedment in mobile robots. He is currently an Assistant Professor with the University of Poitiers, Poitiers, France. His current research interests include concerns mobile robots motion and the interaction with their environment.



Sio-Hoi Ieng received the Ph.D. degree in computer vision from Pierre-and-Marie-Curie University, Paris, France, in 2005.

He was involved in the geometric modeling of noncentral catadioptric vision sensors and their link to the caustic surface. He is currently an Associate Professor with Pierre-and-Marie Curie University and a member of the Vision Institute, Paris. His current research interests include computer vision, with special reference to the understanding of general vision sensors, cameras networks, neuromorphic event-driven vision, and event-based signal processing.

David Filliat received the M.Sc. degree from École Polytechnique, Palaiseau, France, in 1997, and the Ph.D. degree in bioinspired robotics navigation and the Habilitation degree from Pierre-and-Marie-Curie University, Paris, France, in 2001 and 2011, respectively.

He was an expert of robotic programs with the French Armament Procurement Agency for four years. He is currently a Professor with the École Nationale Supérieure de Techniques Avancées (ENSTA) ParisTech, Palaiseau, France. He has been the Head of the Robotics and Computer Vision Team since 2006. He is also a member of the ENSTA ParisTech INRIA FLOWERS Team. His current research interests include perception, navigation, and learning in the frame of the developmental approach for autonomous mobile robotics.

Ryad Benosman is currently a Full Professor with Pierre-and-Marie-Curie University, Paris, France. He leads the Neuromorphic Vision and Natural Computation Group at the Vision Institute, Paris. He focuses mainly on neuromorphic engineering, visual computation, and sensing. He is a pioneer of omnidirectional vision systems, complex perception systems, variant scale sensors, and noncentral sensors. His current research interests include the understanding of the computation operated by the visual system with the goal to establish the link between computational and biological vision.

Prof. Benosman is also invested in applying neuromorphic computation to retina prosthetics, and the Co-Founder of the startup Pixium Vision, Paris.