Low-Latency Event-Based Visual Odometry

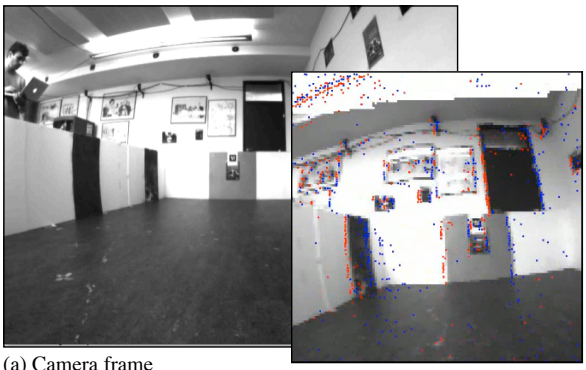
Andrea Censi Davide Scaramuzza

***Abstract*— The agility of a robotic system is ultimately limited by the speed of its processing pipeline. The use of a Dynamic Vision Sensors (DVS), a sensor producing asynchronous events as luminance changes are perceived by its pixels, makes it possible to have a sensing pipeline of a theoretical latency of a few microseconds. However, several challenges must be overcome: a DVS does not provide the grayscale value but only changes in the luminance; and because the output is composed by a sequence of events, traditional frame-based visual odometry methods are not applicable. This paper presents the first visual odometry system based on a DVS plus a normal CMOS camera to provide the absolute brightness values. The two sources of data are automatically spatiotemporally calibrated from logs taken during normal operation. We design a visual odometry method that uses the DVS events to estimate the relative displacement since the previous CMOS frame by processing each event individually. Experiments show that the rotation can be estimated with surprising accuracy, while the translation can be estimated only very noisily, because it produces few events due to very small apparent motion.**

I. Introduction

Perception is still the main problem for high-performance robotics. Once the perception problem is assumed solved, for example by the use of external motion-capture systems, then established control techniques allow for highly performing systems [1]. Such performance, however, is not achievable with onboard sensors such as CMOS cameras [2] or laser rangefinders [3]. The achievable agility of a robotic platform depends on the speed of its processing pipeline (the series of data acquisition plus data processing); more precisely, it is important that the pipeline offers high sampling rate as well as low latency. At the state of the art, the latency of a CMOS-based pipeline is at a minimum in the order of 50-250 ms and the sampling rate is in the order of 15-30 Hz. To obtain more agile robots, we need to switch to faster sensors.

One possible alternative is to use a Dynamic Vision Sensor (DVS) [4]. This is the first commercially available product belonging to a new class of "neuromorphic" sensors [5, 6]. In contrast to a normal CMOS camera, the DVS output is a sequence of asynchronous events rather than regular frames. Each pixel produces an event when the perceived luminance increases and decreases under a certain threshold. This computation is done using an analog circuit, whose biases can be tuned to change the sensitivity of the pixels and other dynamic properties. The events are then timestamped and made available to the application using a digital circuit. Each event is a tuple $\langle t_k, \langle x_k, y_k \rangle, \ p_k \rangle$, where the scalar t_k is the timestamp of the event, the coordinates $\langle x_k, y_k \rangle$ identify the pixel that triggered the event, and the value $p_k \in \{+1, -1\}$ is the event *polarity*, which is the *sign* of the brightness change. DVS-based solutions have been proposed for optic flow reconstruction [7, 8], stereo [9], particle-filter based localization and mapping [10–12], active-landmark-based pose tracking [13].



(a) Camera frame (b) DVS events + grayscale from camera

Fig. 1. We present a visual odometry algorithm that uses low-latency brightness change *events* from a Dynamic Vision Sensor (DVS) and the data from a normal camera to provide the absolute brightness values.

Contribution: In this work, we investigate the use of a DVS for visual odometry, *together* with a regular CMOS camera. The DVS can only sense motion: no events are generated if nothing moves in the scene and the camera is stationary. Furthermore, this sensor has currently a very low resolution: 128×128 pixels. These two limitations might eventually be removed by undergoing research efforts aimed at adding the possibility of sensing static luminance signals and increasing the resolution [14, 15]. In any case, a DVS alone cannot be the only sensor on board of a robot, but rather it needs to be integrated with conventional sensors. We envision an architecture in which agile behavior is obtained by a low-latency control action that uses the data from a sensor like the DVS, while, at slower time-scales, other tasks, such as SLAM, are performed based on slower traditional sensors. Therefore, we try to combine the DVS events with the low frequency data from a normal CMOS camera (hereafter, just "camera") that can provide the absolute brightness value that it is not sensed by the DVS (Fig. 1*b*).

The first challenge is to extrinsically calibrate the two sensors. We devise an unsupervised spatiotemporal calibration technique that is able to create an accurate virtual sensor, notwithstanding several approximations due to the two devices having different resolutions, unsynchronized timestamps, different focus centers for the optical systems, etc. Having created this new "virtual" sensor, which provides low-frequency frames and low-latency events, we design an event-based visual odometry method. Most visual odometry methods are based on detecting and tracking "features" (or "interest

A. Censi is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology. He was supported by NSF/NRI (#1405259) and DARPA/MSEE (#FA8650-11-1-7156). D. Scaramuzza is with the Department of Informatics, University of Zurich. He was supported by the SNSF (project #200021-143607, "Swarm of Flying Cameras") and the National Centre of Competence in Research Robotics.

points") across frames [16]. These approaches cannot be used directly with a DVS, unless the DVS is used to simulate a normal camera, for example by accumulating events in a slice of time and then use it as a "frame" [1] or if the feature tracking problem is defined in the spatiotemporal domain [17–19]. Our goal is to process each event independently as to provide the minimum latency possible.

Outline: Section II describes the camera-to-DVS calibration method. Because we need to use the camera to provide the absolute brightness values that cannot be observed from the DVS alone, calibration in this context means to find for each DVS pixel the corresponding pixel in the camera frame. Calibration is done offline from recorded logs during normal operation of the system (no calibration patterns are necessary).

Section III describes an event-based visual odometry method. We use the DVS events to estimate the relative displacement with respect to the pose where the previous low-frequency frame was acquired. Because the DVS is fast, we can assume that the motion is small, and, thus, decompose the problem along each degree of freedom. Our method is similar in spirit to Markov localization, in the sense that each event is used to define a likelihood function to weigh the current relative motion estimate.

Section IV illustrates experiments for the case of pure rotation. In spite of the various approximations implied by using the camera's data to supply the missing brightness value for the DVS, the system is quite precise and achieves a drift that is small compared with the DVS resolution (1 pixel/°). The system is robust to motion blur. Failures are observed when the majority of the events are due to other objects in the field of view rather than self-motion, and if the rotation is so fast that more than a half field of view moves between successive camera frames.

Section V describes the problem of estimating translation, assuming that the low-frequency images are provided together with the *range image* (either provided natively by a sensor such as a Kinect or estimated using a SLAM algorithm). The algorithm for translation is mostly the same as for rotation, but the results are quite noisy. Our analysis suggests that this is an intrinsic limitation of the DVS: because the sensor only detects changes and has a relatively poor resolution, it cannot be used to estimate translation reliably because translation produces relatively small apparent motion.

Finally, Section VI wraps up and discusses future work.

Notation: We use standard differential geometry notation [20]. $\mathbb{S}^2$ is the unit sphere, the set of vectors in $\mathbb{R}^3$ that have unit length. An element of $\mathbb{S}^2$ is denoted as s and called a *direction.* A direction identifies a pixel of a calibrated camera (we do not use image-space coordinates). An *image*, or *frame*, is a differentiable function $y : \mathbb{S}^2 \to \mathbb{R}$ on the visual sphere. The gradient ∇y is a function on the visual sphere that takes values on the *tangent space* of the sphere: fixed a point $s \in \mathbb{S}^2$, $\nabla y(s)$ is a vector in $\mathbb{R}^3$ that is perpendicular to s. The inner product of two vectors in $\mathbb{R}^3$ is $\langle a, b \rangle$; their cross product is $a \times b$. For a vector $a \in \mathbb{R}^3$, the quantity $a^\times$ is the skew-symmetric 3×3 matrix such that $a^\times b = a \times b$. "Exp" is the matrix exponential and "Log" is the principal logarithm.

[1] Incidentally, this is what we do for visualization purposes (Fig. 1*b*).

II. Spatiotemporal camera/DVS calibration

Assuming a small baseline between the two sensors, we can use the CMOS camera to supply the absolute brightness values in the DVS field of view. The goal of calibration is to find for each DVS pixel the camera pixel that points in the same direction. We describe an offline calibration method that works with logs taken during the system's normal operation, without the need of special calibration patterns or procedures.

The basic idea is that pixels pointing in the same direction see "changes" at the same time, so that a properly defined similarity function captures the spatial relationships between pixels even if they belong to different sensors. The basic principle is thus similar to *calibration by correlation* (see [21] and references therein), but applied to different sensors, and with opportune changes due to the special nature of the DVS data.

The most important assumption of this method is that there is a small baseline between the two sensors; more precisely, the baseline must be negligible with respect to the size of the environment. For example, suppose that (as in our setup) the baseline is $b \simeq 0.01\,\mathrm{m}$ and the minimum distance to an obstacle is $d = 2\,\mathrm{m}$. The direction of the projection of a point of an obstacle on the two sensors can at most differ by $b/d \simeq 0.5°$. In our case this is an acceptable approximation because each DVS pixel covers $\simeq 1°$.

The calibration method has three phases:

1) The first phase consists in a *coarse time synchronization* of the two data sequences. This is needed because the DVS events are timestamped with its own internal clock.
2) The second phase consists in a *coarse spatial calibration*. The output of this phase is an approximate guess for the position of each DVS pixel in the camera frame.
3) The third phase is an *optimization phase* that refines the results of the individual pixels to obtain a globally-consistent solution.

A. Coarse time synchronization

Because the DVS uses its own internal clock and timestamp, the first step needed is the temporal synchronization of the two data streams. If two sensors point (approximately) in the same direction, then they will see (approximately) the same "changes". If we can define two related signals that quantify the changes for the two sensors, then the delay between the two time series is found as the maximum of the cross-correlation between those signals. When large data logs are available, the result is very robust even if the two sensors do not perceive exactly the same part of the scene. The only delicate part is taking care of the fact that the two streams are not equally spaced in time, so that they need to be resampled to a common sampling rate Δ.

For the DVS events, whose timestamps sequence is t_k, for each time t, define the function f_t as the number of events detected in the interval $[t - \Delta, t + \Delta]$ (Fig. 2*a*):

$$f_t = |\{t_k \mid t - \Delta \leq t_k \leq t + \Delta\}|. \tag{1}$$

Let the data from the camera be a sequence of tuples $\langle t_i, y_i \rangle$, where t_i is the timestamp, $y_i : \mathbb{S}^2 \to \mathbb{R}$ is the image (here defined as a function on the visual sphere). Let $\dot{y}_i$ be the discrete approximation to the derivative of the image:

$\dot{y}_i = (y_{i+1} - y_{i-1}/(t_{i+1} - t_{i-1})$. Define the total intensity of the change c_i as the 1-norm of $\dot{y}_i$ over the visual field: $c_i = \int_{\mathbb{S}^2} |\dot{y}_i(s)| \, \mathrm{d}s$. Finally, for each time t, define g_t as the mean intensity for images in the interval $[t - \Delta, t + \Delta]$ (Fig. 2*b*):

$$g_t = \text{mean}\{c_i \mid t - \Delta \leq t_i \leq t + \Delta\}. \tag{2}$$

The correction τ between the two series is the one that minimizes the mismatch between g_t and $f_{t+\tau}$. To obtain both robustness and efficiency, we use a multi-scale analysis, in which we start with a large sampling interval Δ and a large search domain for τ, and then iteratively reducing both.

This method can be used to synchronize other data streams, as long as one can find a signal that correlates with motion and events generations (e.g., angular velocity for the odometry).

B. Coarse spatial calibration

At this point, we assume that the two streams of data are coarsely aligned temporally, and we want to find, for each DVS pixel, a guess of the corresponding pixel for the camera. This will be just a coarse guess that will be refined later.

Let us use the index $a \in A$ to label the DVS pixels, and the index $b \in B$ for the camera pixels. We will define a similarity measure $S(a, b)$, then, for each a, we will choose the corresponding $\hat{b}(a)$ as the pixel b that maximizes the similarity:

$$\hat{b}(a) = \arg\max_{b \in B} S(a, b).$$

In analogy with f_t defined in (1), for each pixel a define the function $f_t(a)$ as the number of events produced observed by that pixel in the interval $[t - \Delta, t + \Delta]$. In analogy with g_t defined in (2), define $g_t(b)$ as the average intensity of the brightness change $\dot{y}$ seen by the camera at pixel b in the interval $[t-\Delta, t+\Delta]$. The similarity $S(a, b)$ is the correlation in time between $f_t(a)$ and $g_t(b)$:

$$S(a, b) = \mathsf{corr}(f_t(a), g_t(b)).$$

This similarity is very simple to implement and can be computed with a streaming algorithm in one pass. There are other possible choices for the similarity involving other signals (such as the intensity of the spatial gradient) and other distances between time series (such as the information distance), though this simplest choice seems to work well enough.

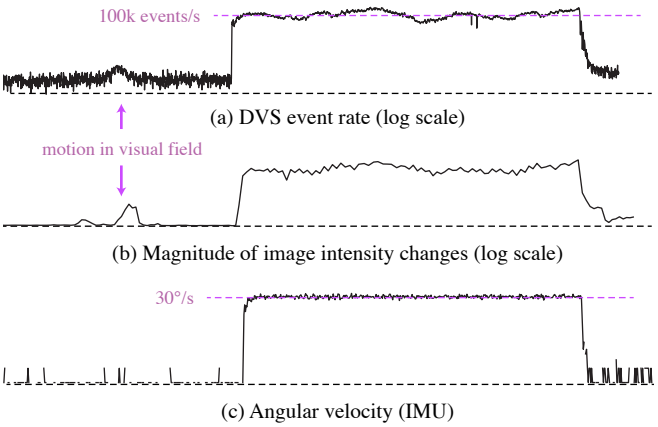
If we computed $S(a, b)$ for each pair of pixel the memory requirements would be prohibitive, because it would be in the order of $\mathcal{O}(\rho_1^2 \rho_2^2)$ where ρ_1, ρ_2 are the resolutions (in pixels/degrees) of the two sensors. Therefore, we compute this similarity using only a subset of the DVS pixels; in our case, we use a 16×16 subgrid of the 128×128 frame.

Fig. 3 shows some examples of the similarity $S(a, b)$, computed on several logs for a total of ~35 minutes. Note that the similarity is ambiguous in the vertical direction. This is due to the properties of the environment, which, in this case, has many vertical features.

It might happen that the DVS pixel has no corresponding point in the camera frame, because it corresponds to pixels that are outside of the frame. In this case the similarity has typically multiple minima (Fig. 3d). A simple confidence measure can quantify this fact. If $\hat{b}(a)$ is the maximum, a confidence level is the ratio of $S(a, \hat{b}(a))$ and the average $S(a, b)$:

$$w^a = S(a, \hat{b}(a)) \,/\, \textstyle\sum_b S(a, b).$$

This confidence measure allows us to distinguish pixels with and without a correspondence by thresholding $w(a)$. The pixels with a low confidence level are marked as invalid and excluded from the successive phases.

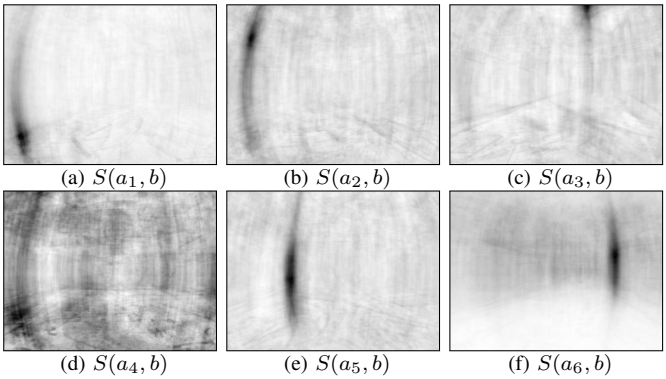


(a) DVS event rate (log scale)

(b) Magnitude of image intensity changes (log scale)

(c) Angular velocity (IMU)

Fig. 2. Synchronization between data streams from different sensors is achieved by computing the offset that maximally superimposes signals that correlate with motion and changes in the visual field.



(a) $S(a_1, b)$ (b) $S(a_2, b)$ (c) $S(a_3, b)$

(d) $S(a_4, b)$ (e) $S(a_5, b)$ (f) $S(a_6, b)$

Fig. 3. A first guess for the position of each DVS pixel a in the image space of the camera is obtained by computing the similarity function $S(a, b)$, here represented in grayscale for four example pixels, among which one that does not have a correspondence.

C. Spatial refinement

Due to noise and ambiguity, the guess obtained independently for each pixel can be rather noisy (Fig. 4a). This last spatial refinement phase is an optimization step that finds a globally consistent solution for all pixels together (Fig. 4b).

Using a standard approach, we define an energy function $E = E_{\mathsf{data}} + \alpha E_{\mathsf{model}}$ that consist of two terms: the data term forces the pixels to have the position previously found, weighted by their confidence level, and the model term encourages the pixels to be in a regular grid. The two terms are weighted by a constant $\alpha > 0$ that quantifies our trust in the model.

1) Data term: This part is best described using image coordinates for the camera frame. For each DVS pixel, we

found the corresponding pixel $\hat{b}(a)$ in the camera frame. Call $\boldsymbol{p}_0^a \in \mathbb{R}^2$ the position of that pixel, and w^a the confidence level. The data term is readily written as

$$E_{\mathsf{data}}(\{\boldsymbol{p}^a\}) = \sum_{a \in A} w^a \|\boldsymbol{p}^a - \boldsymbol{p}_0^a\|^2.$$

2) Model term: The model term should enforce the prior knowledge that we have for the sensor topology. Let us consider the case where the pixels are in a grid, though the procedure can be adapted to different topologies.

For each pixel $a \in A$ let $\mathsf{neig}(a) \subset A$ be the set of the neighbors of a in the grid. Because of the grid assumptions, all neighbors in $\mathsf{neig}(a)$ should be equidistant from a. The constraint is that, for any two neighbors $a_1, a_2 \in \mathsf{neig}(a)$, the points $\boldsymbol{p}^{a_1}$ and $\boldsymbol{p}^{a_2}$ are equidistant from $\boldsymbol{p}^a$:

$$\|\boldsymbol{p}^{a_1} - \boldsymbol{p}^a\| = \|\boldsymbol{p}^{a_2} - \boldsymbol{p}^a\|. \tag{3}$$

We enforce the (soft) constraint by including a penalty to be minimized, such as $|\|\boldsymbol{p}^{a_1} - \boldsymbol{p}^a\|^2 - \|\boldsymbol{p}^{a_2} - \boldsymbol{p}^a\|^2|$. The model term sums the penalty over all pixels and their neighbors:

$$E_{\mathsf{model}}(\{\boldsymbol{p}^a\}) = \sum_{a \in A} \sum_{a_1, a_2 \in \mathsf{neig}(a)} |\|\boldsymbol{p}^{a_1} - \boldsymbol{p}^a\|^2 - \|\boldsymbol{p}^{a_2} - \boldsymbol{p}^a\|^2|.$$

While this penalty function is simple and serves its purpose, it is unclear whether this is the best penalty function for our constraint (3). In general, for any soft constraint to be imposed, one can find an infinite number of penalty functions, and different penalty functions have widely different properties in term of convergence basins, robustness to outliers, etc. [22].

3) Optimization: The energy function just defined is not convex, though it appears to be "convex enough" to be minimized using general-purpose algorithms. Of the standard methods readily available in Numpy, the quasi-Newton BFGS method [23, p. 198] gave the quickest convergence (25 iterations if choosing $\alpha = 0.1$). The results are shown in Fig. 4b.

4) Interpolation: We have so far derived the solution for a subset of the DVS pixels at the corners of a grid. The last step consists in interpolating the result to the rest of the pixels.

The simplest way to interpolate is by using barycentric coordinates. Create a triangulation of the DVS frame using the known grid points. For each pixel whose position is still to be found, the solution is given by averaging the solution of the vertex of the enclosing triangle.

Suppose $\boldsymbol{m} \in \mathbb{R}^2$ are the image coordinates of the pixel in question, and $\boldsymbol{m}_1, \boldsymbol{m}_2, \boldsymbol{m}_3 \in \mathbb{R}^2$ are the coordinates of the vertices. The barycentric coordinates $\lambda_1, \lambda_2, \lambda_3$ are the solution of the equation $\boldsymbol{m} = \lambda_1 m_1 + \lambda_2 m_2 + \lambda_3 m_3$ with the constraint $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

Once the barycentric coordinates have been found using standard formulas[2], the position $\boldsymbol{p}$ is found by interpolating the solutions of the vertices $\boldsymbol{p}^1, \boldsymbol{p}^2, \boldsymbol{p}^3$:

$$\boldsymbol{p} = \lambda_1 \boldsymbol{p}^1 + \lambda_2 \boldsymbol{p}^2 + \lambda_3 \boldsymbol{p}^3.$$

[2]Letting $\boldsymbol{m}_i = \langle x_i, y_i \rangle$ and $\boldsymbol{m} = \langle x, y \rangle$ then

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix}^{-1} \begin{pmatrix} x - x_3 \\ y - y_3 \end{pmatrix}$$

and $\lambda_3 = 1 - \lambda_2 - \lambda_1$. The matrix above is invertible if and only if the three points are not collinear.

III. Event-based rotation estimation

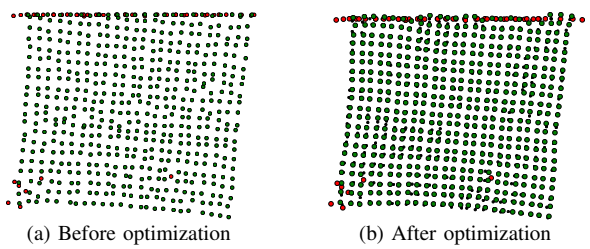
This section describes a visual odometry method that uses the DVS events together with the camera frames. It uses the low-frequency frames as an instantaneous "map" in which to localize based on the events. The method preserves the low-latency nature of the data because each event can be processed individually. Localizing simultaneously with respect to two successive frames allows the method to keep track of the global pose. This section focuses on estimating rotation; Section V discusses translation.

Our design goal of a completely asynchronous event-based method prevents using traditional techniques based on feature tracking. In principle, it is possible to extract features from the events: for example, by accumulating events in a time slice and considering their histogram as an image frame, it is possible to see segment features (Fig. 1*b*). However, this implies accepting additional latency (as events must be accumulated) and is not a robust strategy (for small motions and few events, features cannot be reliably detected).

The basic idea of the method is to estimate the relative motion from the last low-frequency frame, by defining the likelihood that an event can be observed given a certain motion. Because of the sensor's low latency, we can assume that the motion is "small" and we can estimate each degree of freedom independently (i.e., pitch can be estimated independently from yaw and roll) and so we need relatively little computation for processing each event. More in detail, we keep track of a probability distribution relative to the motion for each degree of freedom. Each event can be used to define a likelihood function for the motion that is compatible with seeing that particular event. Clearly, a single event contains very little and ambiguous information, but accumulating the information in a series of events will lead to a precise and robust estimate. In the spirit of Markov localization [24], we can apply a motion model in between events if one is available (i.e. we know the sensor is attached to a robot with known dynamics).

A. Single event vs fixed-event-rate processing

We will describe the method as purely event-based, to highlight that, in principle, it can be implemented asynchronously, perhaps at the hardware level. However, for the current implementations, and the foreseeable future, the computation is done with a *packet* of events, and optimized using vectorization techniques. First, there is a practical constraint: the DVS uses a USB hardware interface and several events are transmitted per packets, though each event has its own hardware-generated timestamp. In addition to this



(a) Before optimization (b) After optimization

Fig. 4. Coordinates of the DVS pixels in the camera's image space, before and after optimization (Section II-C). Red pixels (corresponding to directions outside the camera frame) are marked as invalid and ignored.

constraint, the *micro*second latency that the DVS achieves is unnecessary for robotic platforms: *the goal of estimation is control*, and the time constants of a flying, or wheeled, robot, or plane, or insect, is hardly faster than 1 *milli*second, so there are diminishing returns under that threshold.

Choosing a processing frequency is not straightforward. For a conventional sensor there is a clear trade-off: a high sampling rate corresponds to more precise estimation at the cost of more computation required. With the DVS, the trade-off is different, because the event rate is variable and proportional to the magnitude of the motion; in the limit, without motion, no events are generated. Therefore, processing events at a fixed frequency is either wasteful or inaccurate depending on the situation. A simple heuristics that seems to work well is *fixed event-rate* processing: we accumulate events in packets, and then process them once a given size is reached. For example, in our experimental setup, using the threshold 1000 events/packet corresponds to about 500 Hz processing for 720 °/s rotations. We conjecture that this *fixed-event-rate strategy* uses the least computation for a given expected error.

B. Estimating relative rotation

1) Events and frames: We use the subscript k for the events, and the subcript i for the low-frequency frames. The two series of timestamps are $\{t_i\}_{i\in\mathbb{N}}$ and $\{t_k\}_{k\in\mathbb{N}}$. Neither stream is assumed equispaced in time, and they are not assumed to be accurately synchronized.[3] As previously defined, the frame data is a sequence of tuples $\langle t_i, y_i\rangle$, where $y_i : \mathbb{S}^2 \to \mathbb{R}$ is a function defined on (a subset of) the visual sphere. The event data is a series of tuples $\langle t_k, p_k, s_k\rangle$, where $p_k \in \{+1, -1\}$ is the polarity and $s_k \in \mathbb{S}^2$ is the direction of the pixel of the visual sphere, which we use directly instead of the image coordinates. (Using s_k implies that the sensor is calibrated.)

2) Reference frame: Let us choose convenient reference frames. Suppose that we have received the i-th frame at time t_i and at time $t > t_i$ we want to localize with respect to that frame. If $q_t \in \mathrm{SO}(3)$ is the attitude at time t, we want to recover the relative attitude $q_{t_i}^{-1} q_t$. It is simpler to reformulate everything in relative spatiotemporal coordinates, by letting $q_{t_i} = \mathrm{Id}$ and $t_i = 0$, so that q_t is the relative attitude with respect to the previous image obtained at $t = 0$.

3) Approximating the likelihood for generic rotation: We first derive the formulas for the case of a generic rotation $q_t \in \mathrm{SO}(3)$; later, we will particularize them to the case in which the rotation is constrained to be around a given axis.

The basic observation is that an event must be generated by some spatial gradient in the image—if the image has uniform brightness, no event can be generated by motion. Suppose that at time $t_k \geq 0$ we receive the event $\langle t_k, p_k, s_k\rangle$. We take this as evidence that at time t_k, there was a a nonzero gradient in direction s_k. This implies that, if $q_{t_k} \in \mathrm{SO}(3)$ is the current rotation, there should be a nonzero spatial gradient in the image y_i in direction $q_{t_k}^{-1} s_k$. Therefore, the event s_k is compatible with the current motion being q_{t_k} only if $\nabla y_i(q_{t_k}^{-1} s_k) \neq 0$. A single event cannot disambiguate the motion, therefore we need to accumulate evidence from successive events. This is the basic idea; the next few paragraphs make it more precise.

Formally, we find an approximation to the likelihood $p(\langle t_k, s_k\rangle \mid q_{t_k}, y_i)$. The simplest model of the DVS is that an event is triggered with probability proportional to $|\dot{y}|$. Using the optic flow equation, the derivative $\dot{y}$ is proportional to the angular velocity ω and the gradient ∇y:

$$\dot{y}_t(s) = \langle s \times \nabla y_t(s), \omega_t\rangle = \left\langle \omega^\times s, \nabla y_t(s)\right\rangle. \tag{4}$$

Assuming a small motion of constant velocity ($\omega_t = \omega_0$), the rotation q_t is the matrix exponential of $\omega_o^\times t$: $q_t = \mathrm{Exp}(\omega_o^\times t)$, and, conversely, the velocity ω is the matrix logarithm of q_t:

$$\omega_0^\times = \tfrac{1}{t}\mathrm{Log}(q_t). \tag{5}$$

Combining (4) and (5) we obtain that $\dot{y}_t(s) = \frac{1}{t}\left\langle \mathrm{Log}(q_t)s, \nabla y_t(s)\right\rangle$. Therefore, the likelihood of seeing the event $\langle t_k, p_k, s_k\rangle$ assuming the current pose is q_t is

$$p(\langle t_k, s_k\rangle \,|\, q_{t_k}, y_i) \simeq \tfrac{1}{t_k}\left|\left\langle \mathrm{Log}(q_{t_k}) q_{t_k}^{-1} s_k, \nabla y_i(q_{t_k}^{-1} s_k)\right\rangle\right|. \tag{6}$$

Once we have defined this likelihood function we can implement a Bayesian filter. However, it would be terribly inefficient to iterate over the entire space $\mathrm{SO}(3)$ for each event. Assuming small motion, we can decompose the problem over each degree of freedom.

4) Simplification for one degree of freedom: Fix a rotation axis described by a unit vector $u \in \mathbb{S}^2$. Assuming that the sensor only rotates along u, we can write the rotation q_t as a function of the angle θ_t along that axis:

$$q_t = \mathrm{Exp}(\theta_t u^\times). \tag{7}$$

The angle θ_t is going to be the parametrization for the rotation. Rewriting (6) using (7), we obtain that $p(\langle t_k, p_k\rangle \mid \theta_{t_k}, y_i) \simeq$

$$\begin{aligned}
&\tfrac{1}{t_k}\left|\left\langle \mathrm{Log}(q_{t_k}) q_{t_k}^{-1} s_k, \nabla y_i(q_{t_k}^{-1} s_k)\right\rangle\right| \\
= \quad &\{\text{Using (7)}\} \\
&\tfrac{1}{t_k}\left|\left\langle \mathrm{Log}(\mathrm{Exp}(\theta_t u^\times))\mathrm{Exp}(\theta_t u^\times)^{-1} s_k, \nabla y_i(\mathrm{Exp}(\theta_t u^\times)^{-1} s_k)\right\rangle\right| \\
= \quad &\{\mathrm{Log}(\mathrm{Exp}(x)) = x, \text{ for small } x\} \\
&\tfrac{1}{t_k}\left|\left\langle \theta_t u^\times \mathrm{Exp}(\theta_t u^\times)^{-1} s_k, \nabla y_i(\mathrm{Exp}(\theta_t u^\times)^{-1} s_k)\right\rangle\right| \\
= \quad &\{\mathrm{Exp}(u^\times)^{-1} = \mathrm{Exp}(-u^\times)\} \\
&\tfrac{1}{t_k}\left|\left\langle \theta_t u^\times \mathrm{Exp}(-\theta_t u^\times) s_k, \nabla y_i(\mathrm{Exp}(-\theta_t u^\times) s_k)\right\rangle\right| \\
= \quad &\{\langle a^\times b, c\rangle = -b^T a^\times c\} \\
&\tfrac{|\theta_t|}{t_k}\left|\left\langle \mathrm{Exp}(-\theta_t u^\times) s_k, u^\times \nabla y_i(\mathrm{Exp}(-\theta_t u^\times) s_k)\right\rangle\right|.
\end{aligned} \tag{8}$$

Equation (8) can be rewritten in a simpler form. Because $\mathrm{Exp}(-\theta_t u^\times) s_k$ appears twice, we refactor this term as the function $\varphi^u(\theta, s)$ that rotates s of $-\theta$ around the axis u:

$$\begin{aligned}
\varphi^u : \mathbb{S}^1 \times \mathbb{S}^2 &\to \mathbb{S}^2, \\
\theta, s &\mapsto \mathrm{Exp}(-\theta u^\times) s.
\end{aligned}$$

To further simplify (8), define the function $\Psi_y^u(x)$ as

$$\begin{aligned}
\Psi_y^u : \mathbb{S}^2 &\to \mathbb{R}, \\
x &\mapsto \left\langle x, u^\times \nabla y_i(x)\right\rangle.
\end{aligned} \tag{9}$$

With this compact notation, the likelihood approximation is

$$p(\langle t_k, s_k\rangle \mid \theta_{t_k}, y_i) = \tfrac{|\theta_t|}{t_k} |\Psi_y^u(\varphi^u(\theta_t, s_k))|. \tag{10}$$

[3] In our particular experimental setup, we discovered we had to work around this problem because the images were timestamped by ROS, with considerable userspace jitter, rather than directly at the driver or hardware level. Because this seems to be a likely common problem, we set as a design goal to not rely on precise synchronization of frames and events.

The function Ψ_y^u gives all that is needed to know about the image y_i: it is the intensity of the gradient that counts towards generating spike events for the direction of interest u. This function needs to be computed only once per image, and, with further manipulation, it can be written as the absolute value of a linear combination of the gradients in image space.

The term $|\theta_t|/t_k$ in (10) is the absolute value of the angular velocity: with a change of variable, we could express everything using the angular velocity, but using the relative rotation gives a more intuitive formulation overall.

The term $\varphi(\theta_t, s_k)$, seen as a function of θ_t, describes an arc in the visual sphere. Crucially, this is the only quantity that depends on the event, through the direction s_k. Algorithmically, this implies that, given an event in direction s_k, we need to iterate over θ, and trace an arc in image space, and the likelihood for θ is given by evaluating Ψ_y^u at $\varphi(\theta_t, s_k)$.

C. Filtering and tracking

Tracking the relative motion with respect to the last received frame is an instance of a Bayesian filter:

1) *Initialization:* Wait for y_i. Compute $\Psi_{y_i}^u$. Set initial distribution for $p(\theta_0)$ to a uniform distribution.
2) For each new event $\langle t_k, p_k, s_k \rangle$ received:
   a) *Prediction:* Evolve the probability distribution using a motion model $p(\theta_{t_k} \mid \theta_{t_{k-1}})$.
   b) *Update:* Weight the probability distribution using the likelihood given by (10).

As mentioned before, while we could consider each event individually, it might be preferable to process events packets, and optimize the evaluation of (10) using vectorized operations.

The last block that we need for a complete algorithm is defining what happens when a new frame y_{i+1} is received. To keep track of the global pose, we need to find the relative pose between y_i and y_{i+1}. This can be done by running the localization filter above independently for the two images concurrently; once the DVS is localized with respect to both images, the relative motion between the two images can be estimated, a global state variable can be updated with the global pose, and the oldest image discarded.

D. Further refinements

1) Using the polarity information: We can use the event polarity p_k to improve the likelihood approximation (6). The constraint is that the polarity p_k must be the sign of the brightness variation $\dot{y}$. Starting from (4) and redoing similar steps as before, we obtain that $\mathsf{sgn}(\dot{y}) = \mathsf{sgn}(\theta_t \Psi_y(\varphi(\theta_t, s_k)))$. Thus, in absence of noise, the position θ_{t_k} is compatible with the event only if

$$p_k = \mathsf{sgn}(\theta_t \Psi_y(\varphi(\theta_t, s_k))). \tag{11}$$

To take into account noise and other unmodelled phenomena (such as objects that move of independent motion, thus not conforming to (4)) we soften this constraint. The constraint is respected if the product $p_k(\theta_t \Psi_y(\varphi(\theta_t, s_k)))$ is positive, thus a robust likelihood function can be compactly written as

$$p(\langle t_k, s_k, p_k \rangle \mid \theta_{t_k}, y_i) = H(p_k \tfrac{\theta_t}{t_k} | \Psi_y(\varphi(\theta_t, s_k))), \tag{12}$$

where $H : \mathbb{R} \to \mathbb{R}^+$ is a function that implements a soft threshold, such as $H(x) = c + x$ for $x > 0$, and $H(x) = c > 0$ for $x < 0$. The constant c depends on the noise level. (An interesting extension would be learning H from data.)

2) Subpixel maxima detection: Finally, we mention a simple trick regarding how to compute the maximum of the probability distribution of θ when an unimodal output is needed, for visualization (as in the experiment section) or for further processing. The resolution of the probability distribution for θ is the resolution of the sensor, which is 1 pixel/° for our setup. If we only extract the maximum of the buffer ($\hat{\theta}_t = \arg\max p(\theta_t)$), then the answer suffers from a discretization of 1 pixel/°. A better approach is to fit a local Gaussian-plus-constant approximation to the distribution, and use the mean of the fitted Gaussian as the unimodal guess.

IV. Experiments

We set up the DVS + camera system on a TurtleBot. The robot has, in theory, only two degrees of freedom (rotation on the yaw axis, and forward translation), due to the nonholonomic dynamics. However, in practice, it is quite unstable on its two wheels, so that it wobbles back and forth when it starts a translational motion—and our method does detect this oscillatory pitch movement.

For the quantitative evaluation we focus on yaw, so that we can use the robot's odometry for evaluating the precision. In the following the packet size is set to 1000 events/packet.

In general, in our partially engineered test environment rich in horizontal gradient (Fig. 5*a*) the method is quite robust, and the only failures (where the maxima of the distribution is far from the ground truth) happen when most of the field of view is covered by dynamic objects (Fig. 5*c*). Dynamic objects bring two kinds of nuisances: first, they create additional confusing events in the DVS data; second, they are not good references in the camera frames.

The drift, as evaluated with the odometry, is in the order of 1% of the total rotation. This is measured for slow motions, as for fast motions the odometry tends to be unreliable, as there is wheel slippage when the robot makes abrupt motions (Fig. 6). Rotation estimation is robust even in the presence of translation (Fig. 7).

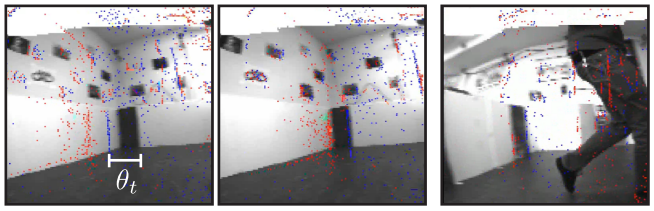
Choosing slightly different event packet sizes does not change the estimate much (Fig. 8), except that choosing 500 events/packets (Fig. 8*a*) instead of 2000 (Fig. 8*c*) gives a 4 times more frequent update.

The precision that we achieve is much smaller than the sensor resolution, thanks to the subpixel trick explained in Section III-D.2. The effect can be clearly seen for the data in Fig. 8*d,* where it is not used, and the discretization corresponds to the sensor resolution.

V. Estimating translation

We were not able to produce consistently precise estimates of translational motion. Nonetheless, for completeness, we describe what we tried and the partial success we obtained.

The basic problem for estimating translation is that in the robot localization scenario the apparent motion due to translation is much smaller that the apparent motion due to rotation. Suppose the robot moves forward at velocity $v = 1$ m/s. The largest apparent motion is if an object is orthogonal to the motion vector; as for the distance, suppose that the robot keeps a safe distance of $d = 2\,m$ from the environment.



(a) Frame and events. (b) Events re-aligned given the estimate. (c) Failure due to moving objects.

Fig. 5. Panel *a* shows a typical input, with the events superimposed with the grayscale image. Panel *b* shows the same events artificially rotated by the estimated θ_t. Panel *c* shows a quasi-failure state, in which moving objects do not provide a reliable reference image to match the events.

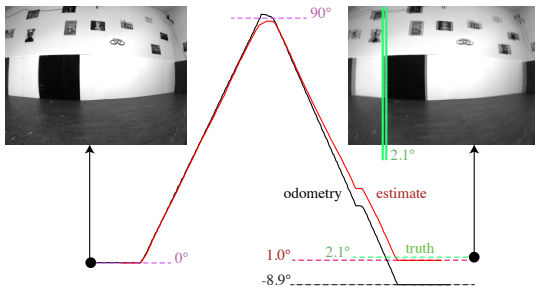
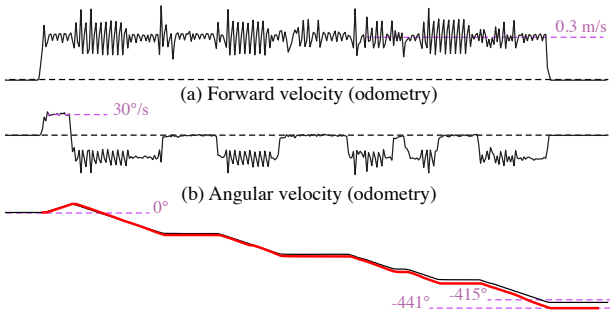
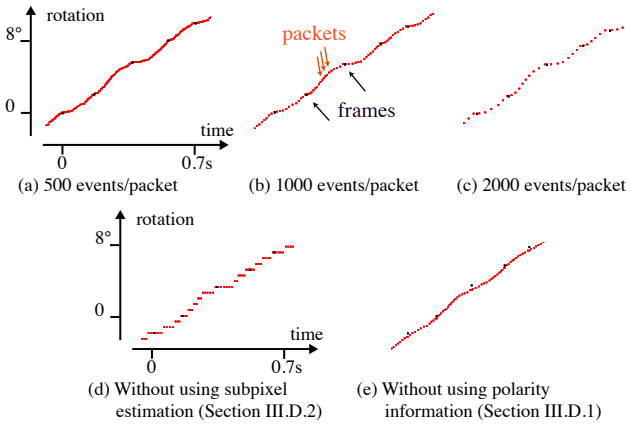


Fig. 6. Odometry (black) vs estimate (red) compared with ground truth. The estimate drift is 1° after a 180° rotation (~0.005%). In this case, the odometry drift is larger (8.9°) because of wheels slippage.



(a) Forward velocity (odometry)
(b) Angular velocity (odometry)
(c) Odometry heading (black) and estimated heading (red)

Fig. 7. Rotation estimation is robust even in the presence of translation.



(a) 500 events/packet (b) 1000 events/packet (c) 2000 events/packet
(d) Without using subpixel estimation (Section III.D.2) (e) Without using polarity information (Section III.D.1)

Fig. 8. Effect of changing different parameters. The figure show the low-latency heading estimation in the first part of the trajectory shown in Fig. 6. Black dots mark the estimation corresponding to the low-frequency frames. The "wobbliness" is due to the imprecise timestamp of the frames. Panels *a, b, c* show the effect of changing the packet size (500, 1000, and 2000 events/packets, respectively). Panel *d* shows the effect of not using subpixel maxima extraction (Section III-D.2): the discretization corresponds to the sensor resolution. Finally, panel *e* shows the effect of not using the polarity information (Section III-D.1).

Then the apparent motion is at most $v/d = 0.5\,\text{rad/s} = 30°/s$. For a pixel spread of 1°, it takes $1°/30°/s = 0.03s$ to have an apparent motion of 1 pixel. This is one order of magnitude less of what we expect for rotation; a rotation rate of $360°/s$ is easily achieved. We conclude that the DVS with its current spatial resolution can reliably detect translational motion only at frame rates, for which a normal camera would suffice.

This suggests that in a robot architecture the DVS should be used only for estimating and compensating fast rotations, while estimation of translation should be done more using traditional sensor and visual odometry pipelines. Nevertheless, we discuss how translation can be recovered by adapting the method previously described.

A. Variation of the rotation estimation procedure

For recovering translation, we also need to know the depth data attached to the luminance data. This can be obtained from the sensor itself (e.g., for a Kinect) or estimated using a visual odometry technique. Let $d_i : \mathbb{S}^2 \to \mathbb{R}^+$ be the range image attached to the intensity image y_i.

For the purpose of estimating translation, we can assume that rotation has already been estimated and compensated. (It is easy to compensate the rotation by simply rotating the DVS events, as in Fig. 5*a–b*.) The analogous relation to (4) for translational motion with velocity $v_t \in \mathbb{R}^3$ is

$$\dot{y}_t(s) = \tfrac{1}{d_i(s)} \langle \nabla y_t(s), v_t \rangle . \tag{13}$$

As before, assume we are estimating the translation along one axis, so that the relative position with respect to the previous image is $x_t u$ where $u \in \mathbb{S}^2$ is a fixed direction and x_t is a scalar coordinate (analogous of θ_t). We can redo all passages that led to (10), obtaining a similar result for the likelihood:

$$p(\langle t_k, s_k \rangle \mid x_{t_k}, y_i) = \tfrac{|x_{t_k}|}{t_k} |\Psi_y(\gamma_d^u(x_t, s_k))|. \tag{14}$$

Instead of the function φ^u, we have an analogous function $\gamma_d^u(x, s)$ that describes the apparent motion of a feature point in direction s under the translation x in direction u:

$$\begin{aligned} \gamma_d^u : \mathbb{R} \times \mathbb{S}^2 &\to \mathbb{S}^2, \\ x, s &\mapsto (d_i(s)s - xu)/\|d_i(s)s - xu\|. \end{aligned}$$

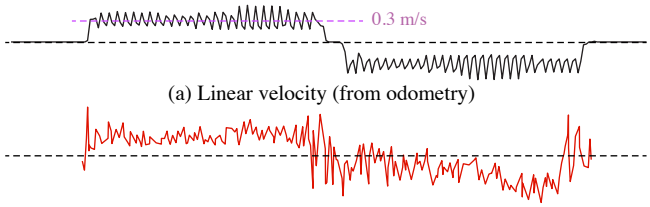
Based on this equation we can adapt the algorithm described in Section III with the simple substitution of φ^u with γ_d^u.

In our setup, this method gave essentially unusable results because of the extremely small apparent motion. However, it might prove useful in situations of low angular velocity and high translational velocity, such as the automotive scenario.

B. Estimating instantaneous translational velocity

The methods presented so far assumed that the motion was small but finite, in the sense that θ and x are small but nonzero. An alternative is to directly make an infinitesimal motion assumption; in the sense of assuming $x = 0$ and estimating the velocity $v = \dot{x}$.

The optic flow equation (13), assuming that we know y, d, and $\dot{y}$, is a linear equation in the unknown v. This relation is usually useless in frame-based processing, because the derivative $\dot{y}_t(s)$ must be obtained by discrete difference of successive images, thus losing the infinitesimal motion assumptions; however, in our scenario, we can approximate



(a) Linear velocity (from odometry)

(b) Estimated linear velocity from DVS data

Fig. 9. Estimation of translation velocity using the method in Section V-B gives mostly the correct sign, but overall a very noisy signal. This data was computed using a constant-depth assumption, in a scenario where this hypothesis is mostly correct.

the derivative $\dot{y}$ by *accumulating* the *events*, rather than *differentiating* the *frames*.

Experimentally, we find that we are able to estimate reliably the *sign of the velocity* but the estimate is quite noisy (Fig. 9). Apart from the small apparent motion problem mentioned before, another factor that might become relevant is that the DVS (from which we estimate $\dot{y}$) and the camera (from which we estimate ∇y) are not exactly sampling the exact light field as they have different sensitivity; at the very least, we need to allow for a nonlinear scaling of the intensities. Further development of the sensor to provide also a grayscale signal might improve these results [14, 15].

VI. Conclusions

This paper described a low-latency visual odometry method based on the fusion of the events data from a Dynamic Vision Sensor (DVS) and the absolute brightness levels provided by a normal CMOS camera. The two sensors are automatically spatiotemporally calibrated, based on the computation of similarity statistics from logs obtained from the normal operation of the system. Our visual odometry method preserves the low-latency nature of the data, as the information from each event can be processed individually to update the current estimate. We showed that, in spite of the high spatial discretization of the sensor, and the various approximations implied from the fusion of the two sensors' data, our method recovers fast rotations with a relatively small drift. Translation cannot be reliably estimated, due to the small apparent motion that produces in a typical robotic setting. This suggests to use the DVS for estimating and compensating fast rotational motion, and estimate translation with a conventional architecture.

Future algorithmic work consists in integrating this method in a complete SLAM system. There is much theoretical work to do as well. We have justified our conclusions regarding the precision achievable using back-of-the-envelope calculations, but it would be desirable to have precise formulas that give the achievable accuracy from a mathematical model of the DVS events generation process as well as intrinsic/extrinsic parameters. Similarly, it would be interesting to have a solid mathematical justification of the advantages of fixed-event-rate processing as opposed to other strategies.

Acknowledgments: Thanks to Jonas Strubel, Matia Pizzoli, and Christian Forster for their help in capturing the data, and to Carolin Baez for careful proofreading.

References

[1] S. Lupashin and R. D'Andrea. "Adaptive fast open-loop maneuvers for quadrocopters". In: *Auton. Robots* 33.1-2 (2012). DOI: 10.1007/s10514-012-9289-9.

[2] S. Weiss, D. Scaramuzza, and R. Siegwart. "Monocular-SLAM-based navigation for autonomous micro helicopters in GPS-denied environments". In: *Journal of Field Robotics* 28.6 (2011). ISSN: 1556-4967. DOI: 10.1002/rob.20412.

[3] S. Shen, N. Michael, and V. Kumar. "Autonomous multi-floor indoor navigation with a computationally constrained MAV". In: *Int. Conf. on Robotics and Automation*. 2011. DOI: 10.1109/ICRA.2011.5980357.

[4] P. Lichtsteiner, C. Posch, and T. Delbruck. "A 128 × 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor". In: *IEEE J. of Solid-State Circuits* 43.2 (2008). DOI: 10.1109/JSSC.2007.914337.

[5] S.-C. Liu and T. Delbruck. "Neuromorphic sensory systems". In: *Current Opinion in Neurobiology* (2010). DOI: 10.1016/j.conb.2010.03.007.

[6] T Delbruck, B Linares-Barranco, E Culurciello, and C Posch. "Activity-driven, event-based vision sensors". In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. Paris, France, 2010. ISBN: 1424453089. DOI: 10.1109/ISCAS.2010.5537149.

[7] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. Srinivasan. "Asynchronous Frameless Event-based Optical Flow". In: *Neural Networks* 27 (2012). DOI: 10.1016/j.neunet.2011.11.001.

[8] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi. "Event-Based Visual Flow". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.2 (2014). DOI: 10.1109/TNNLS.2013.2273537.

[9] P. Rogister, R. Benosman, S.-H. Ieng, P. Lichtsteiner, and T. Delbruck. "Asynchronous Event-Based Binocular Stereo Matching". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.2 (2012). ISSN: 2162-237X. DOI: 10.1109/TNNLS.2011.2180025.

[10] D. Weikersdorfer and J. Conradt. "Event-based particle filtering for robot self-localization". In: *Int. Conf. on Robotics and Biomimetics*. 2012. DOI: 10.1109/ROBIO.2012.6491077.

[11] D. Weikersdorfer, R. Hoffmann, and J. Conradt. "Simultaneous Localization and Mapping for Event-Based Vision Systems". In: *Computer Vision Systems*. Ed. by M. Chen, B. Leibe, and B. Neumann. Vol. 7963. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-39402-7_14.

[12] R. Hoffmann, D. Weikersdorfer, and J. Conradt. "Autonomous indoor exploration with an event-based visual SLAM system". In: *Europ. Conf. on Mobile Robots*. 2013. DOI: 10.1109/ECMR.2013.6698817.

[13] A. Censi, J. Strubel, C. Brandli, T. Delbruck, and D. Scaramuzza. "Low-latency localization by Active LED Markers tracking using a Dynamic Vision Sensor". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Tokyo,Japan, 2013. DOI: 10.1109/IROS.2013.6696456.

[14] 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". In: *IEEE Journal of Solid-State Circuits* 46.1 (2011). DOI: 10.1109/JSSC.2010.2085952.

[15] R. Berner, C. Brandli, M. Yang, S.-C. Liu, and T. Delbruck. "A 240 × 180 120db 10mw 12μs-latency sparse output vision sensor for mobile applications". In: *Int. Image Sensor Workshop (IISW)*. 2013.

[16] D. Scaramuzza and F. Fraundorfer. "Visual Odometry [Tutorial]". In: *Robotics and Automation Magazine* 18.4 (2011). DOI: 10.1109/MRA.2011.943233.

[17] T. Delbruck and P. Lichtsteiner. "Fast sensory motor control based on event-based hybrid neuromorphic-procedural system". In: *IEEE International Symposium on Circuits and Systems*. 2007. DOI: 10.1109/ISCAS.2007.378038.

[18] J Conradt, M. Cook, R Berner, P Lichtsteiner, R. J. Douglas, and T Delbruck. "A pencil balancing robot using a pair of AER dynamic vision sensors". In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. 2009. DOI: 10.1109/ISCAS.2009.5117867.

[19] M. Hofstetter. "Temporal Pattern-Based Active Marker Identification and Tracking Using a Dynamic Vision Sensor". Master thesis. Institue of Neuroinformatics, ETH Zurich, 2012.

[20] M. do Carmo. *Riemannian Geometry*. Birkhauser, 1994.

[21] A. Censi and D. Scaramuzza. "Calibration by correlation using metric embedding from non-metric similarities". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2013). DOI: 10.1109/TPAMI.2013.34.

[22] P. de la Puente. "Probabilistic mapping with mobile robots in structured environments". PhD thesis. Universidad Politécnica de Madrid—ETSI Industriales, 2012.

[23] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.

[24] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005. ISBN: 0262201623.