UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Industriale DII

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Using machine learned features for robot ego-motion estimation through an event-camera

Relatore:
Ing. Sebastiano Chiodini

Trevisanuto Giovanni, 2021383

Anno Accademico 2022/2023

Abstract

This thesis analyses the advantages offered by event-cameras in ego-motion estimation. Traditional cameras suffer from poor performance in low light conditions or high-speed motion. Event-cameras overcome these limitations by detecting and processing only the changes in the visual scene, offering a higher dynamic range and a lower power consumption. In particular, this thesis analyses a feature detection method based on machine learning that takes advantage of the peculiarities of this type of data, resulting in higher precision and longer feature tracks with respect to handcrafted methods. The inference pipeline is composed of a module repeated twice in sequence, formed by a Squeeze-and-Excite block and a ConvLSTM block with residual connection. It is followed by a final convolutional layer that provides the trajectories of the corners as a sequence of heatmaps. A novel training method is described and evaluated.

Riassunto esteso

Questa tesi analizza i vantaggi offerti dalle event-camera nella stima dell'ego-motion. Le telecamere tradizionali presentano limitazioni importanti in condizioni di scarsa luminosità o di di movimento ad alta velocità. Le event-camera superano queste limitazioni rilevando e processando solo i cambiamenti nella scena in maniera completamente asincrona, offrendo un range dinamico più elevato e un consumo energetico inferiore. In particolare, questa tesi analizza un metodo di rilevamento dei keypoints basato sul machine learning che sfrutta le peculiarità di questo tipo di dati, ottenendo una maggiore precisione e tracks più lunghi rispetto ai metodi tradizionali. La pipeline di inferenza è composta da un modulo ripetuto due volte in sequenza, costituito da un blocco Squeeze-and-Excite e da un blocco ConvLSTM con residual connection. Questi sono poi seguiti da un layer convoluzionale finale che fornisce le traiettorie dei keypoints come una sequenza di heatmaps. Da queste è poi possibile risalire alle posizioni dei keypoints identificando i massimi locali che superano una certa soglia. Viene descritto e valutato un nuovo metodo per il training della rete, basato su dati acquisiti tramite una camera DAVIS 346, che permette l'acquisizione contemporanea di frame e eventi. I dati di training sono costruiti utilizzando il metodo di Harris nei frame tradizonali e interpolando linearmente la posizione dei keypoint trovati tra ciascuna coppia di immagni consecutive per ottenere la posizione con la risoluzione temporale (molto maggiore) richiesta dagli eventi. Queste label sono poi confrontate con le heatmap fornite in output dalla rete. In questo modo si incoraggia la corrispondenza tra i masismi relativi delle heatmap e la posizione dei keypoints. I keypoint ottenuti sono quindi sottoposti a matching tra "frame" successivi, filtrando quelli affetti da rumore tramite un algoritmo RANSAC, e tracciati nel tempo. In particolare si è utilizzato un semplice algortimo Nearest Neighbor, scelta resa possibile dall'elevatissima risoluzione temporale della rilevazione. Le tracce ottenute, pur essendo tra loro coerenti e rappresentando con buona approssimazione il movimento registrato, sono affette da un'importante quantità di rumore. Questo è probabilmente dovuto alla qualità dei dati di training, che in diversi casi includono keypoint non rilevati dal metodo di Harris, generando instabilità nei risultati forniti dalla rete. Inoltre, parte di questo rumore potrebbe essere rimossa tramite l'utilizzo di un algoritmo di tracciamento più sofisticato.

Contents

List of Figures

List of Tables

Ego-motion estimation

Ego-motion estimation refers to the process of estimating the motion of a camera or an observer in a given environment based on visual information captured by the camera. It is a fundamental task in robotics, augmented reality, autonomous vehicles, and other computer vision applications. Ego-motion estimation enables the tracking of camera movements, such as translation (movement along the x, y, and z axes) and rotation (yaw, pitch, and roll). In particular, Monocular Visual Odometry (VO) is a technique that estimates the ego-motion of a camera using a single monocular camera. It relies on the analysis of consecutive frames to track visual features and estimate camera motion.
A high-level overview of the typical pipeline for monocular VO includes:

1. Feature Detection and Tracking: In the first step, distinctive visual features, such as corners or keypoints, are detected in the initial frame. Common feature detectors include Harris, FAST or SURF. These features are then tracked across subsequent frames to serve as reference points for estimating camera motion.

2. Feature Matching: In each frame, features are matched between the current frame and the previous frame to establish correspondences. Various techniques can be used for feature matching, such as nearest-neighbor matching or robust feature matching methods. The matched feature correspondences provide information about the apparent motion of the camera.

3. Motion Estimation: Using the matched feature correspondences, the camera motion is estimated. The essential matrix is computed from the feature correspondences, and robust estimation methods like RANSAC (RAndom SAmple Consensus) are used to filter out outliers and obtain an accurate estimate of the essential matrix. From the essential matrix, the relative camera pose (rotation and translation) between frames can be extracted.

4. Scale Estimation: Monocular VO cannot directly determine scale because it lacks depth information. However, scale can be estimated by incorporating additional information sources. For example, scale can be inferred from known object sizes in the scene, or by using data from other sensors such as IMU (Inertial Measurement Unit) or GPS.

5. Trajectory Integration: The estimated camera motion is integrated over time to generate the camera trajectory. By accumulating the relative motion estimates, a continuous estimate of the camera's 3D position and orientation is obtained.

6. Loop Closure: In long-duration monocular VO, errors can accumulate over time, leading to drift. Loop closure techniques aim at recognizing previously

visited locations and correcting the accumulated drift. Loop closure can be achieved by matching features between distant frames or by utilizing place recognition techniques.

In particular, this work focuses on feature detection, matching and tracking, in hopes to help developing a full event-based VO pipeline.

1.1 Camera calibration

The calibration operation is a necessary step in all the applications that require a camera, as it allows correcting the distortions, both radial and tangential, introduced by camera by extracting its intrinsic parameters. The behavior of a camera can be described by

$$k \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S_x f & S_\theta f & x_0 \\ 0 & S_y f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

where $[X, Y, Z]^T$ is the vector containing the coordinates of the generic observed point in the observer's reference frame, $[x, y, z]^T$ is the vector containing the coordinates of its projection on the image plane, S_x and S_y represent the pixel density respectively along the x and y axes, S_θ is a *skew factor* that takes into account the non-orthogonality between the x and y axes of the camera's imaging system and x_0 and y_0 are the coordinates of the top left corner of the image in the image plane reference frame.
In this work, the Zhang calibration method was used, which involves capturing a sequence of images of a chessboard of known dimensions in different positions. The dedicated software (in this specific case, the calibration module of DV [15] by iniVation) processes all the images and, with an error minimization routine, provides the desired values. Specifically, if the scene's coordinate system origin is centered at the top-left corner of the chessboard and the Z-axis is chosen to be orthogonal to it, the coordinates of the chessboard's square vertices can be completely determined based on the known dimensions of the chessboard. The chessboard's vertices are identified in the images using feature detection techniques. Using a chessboard with m vertices, m systems of linear equations can be written as follows

$$k\vec{m} = k \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,4} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = H\vec{w}'$$

from which follows

$$k\vec{m} = H\vec{w}' \implies k\vec{m} \times H\vec{w}' = 0 \implies \begin{bmatrix} yh_3^T\vec{w}' - h_2^T\vec{w}' \\ h_1^T\vec{w}' - xh_3^T\vec{w}' \\ xh_2^T\vec{w}' - yh_1^T\vec{w}' \end{bmatrix} = 0, \; where \; H = \begin{bmatrix} h_1^T \\ h_2^T \\ h_3^T \end{bmatrix}$$

which can be written as

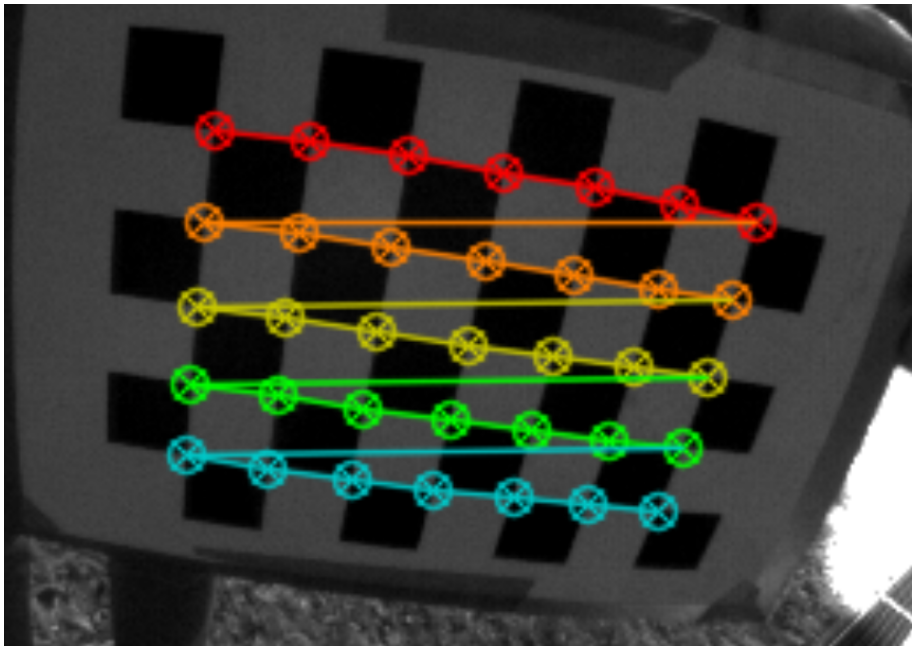


Figure 1.1: Calibration process using a checkerboard pattern.

$$\begin{bmatrix} 0 & -\vec{w}'^T & y\vec{w}'^T \\ \vec{w}'^T & 0 & -x\vec{w}'^T \\ -y\vec{w}'^T & x\vec{w}'^T & 0 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = Ah = 0$$

For each chessboard vertex, 3 equations in 9 unknowns can be written, but only 2 of them are linearly independent. Therefore, for each image, there will be $2m$ equations in 9 unknowns. An initial estimation of H_i is obtained by minimizing the norm of Ah while imposing $||h|| = 1$. Starting from this initial estimation, the error is further minimized by comparing the predicted coordinates of each chessboard vertex with the measured ones. More precisely, the equation $\sum_j ||m_j - H_i w'_j||^2$ is minimized using a least squares algorithm.
Once H_i is known (up to a scale factor), it is possible to derive the intrinsic and extrinsic parameters of the camera considering the following relationship

$$H = kA \begin{bmatrix} r_1 & r_2 & T \end{bmatrix}$$

where A represents the intrinsics matrix. Considering R is an orthogonal matrix, we can write

$$r_1^T r_2 = 0 \implies h_1^T A^{-T} A^{-1} h_2 = 0$$

$$r_1^T r_1 = r_2^T r_2 \implies h_1^T A^{-T} A^{-1} h_1 = h_2^T A^{-T} A^{-1} h_2$$

The unknowns are therefore the elements of the matrix $B = A^{-T}A^{-1}$, which, being A an upper triangular matrix, is symmetric. This results in a linear system of $2n$ equations in 6 unknowns, solvable if at least 3 images are available. At this point, it is possible to estimate the distortion coefficients and, if needed, proceed with image rectification. In this work, MATLAB's *undistortImage* function was used.

1.2 Motion estimation

While feature detection, matching and tracking are to be discussed in the rest of this work, a brief overview of motion estimation is given in this section.
Once the feature correspondences are established between consecutive frames, the camera motion can be estimated. One commonly used approach is to compute the essential matrix from the feature correspondences. The essential matrix represents the geometric relationship between two camera views and encodes the relative camera pose. It describes how corresponding points in one image relate to corresponding points in another image, considering both rotation and translation. The essential matrix is a 3×3 matrix that relates points in the two camera views using epipolar geometry.
To compute the essential matrix, a robust estimation method like RANSAC (RAndom SAmple Consensus) is often employed. RANSAC is an iterative algorithm that filters out outliers and provides a more accurate estimation of the essential matrix. It randomly samples a minimal set of correspondences, calculates the essential matrix for each sample, and then evaluates the consensus between the estimated models and the remaining correspondences. The process is repeated multiple times, and the model with the highest consensus is selected as the best estimate.
Once the essential matrix is obtained, the relative camera pose between frames can be extracted. The essential matrix decomposes into the product of a rotation matrix and a translation vector, representing the camera's rotational and translational displacement. This decomposition can be achieved using techniques like singular value decomposition (SVD) or eigenvalue decomposition.
By continuously integrating the relative motion estimates over multiple frames, a trajectory is formed, representing the camera's path through space over time. The accumulated camera poses provide a continuous estimate of the camera's 3D position and orientation at each point in time.
It's important to note that trajectory integration is subject to accumulation errors and drift over time. Errors in motion estimation, camera calibration, and environmental factors can cause deviations from the true trajectory. To mitigate these errors, techniques such as loop closure detection, bundle adjustment, and sensor fusion with additional sensors (e.g., IMU or GPS) can be employed to refine the trajectory and improve accuracy.

Event-cameras and event processing

In recent years, the field of computer vision has witnessed a significant breakthrough with the emergence of event-based cameras. Unlike traditional frame-based cameras, event-based cameras capture visual information based on changes occurring in the scene rather than capturing images at fixed time intervals. This novel technology has revolutionized vision sensing by enabling high-speed, low-latency and efficient visual processing for various applications.
Event-based cameras are also known as neuromorphic or asynchronous cameras and are inspired by the functioning of the human eye and the brain's visual processing system. Their natural application, and the way to exploit their full potential, is to couple them with neuromorphic processors that handle data through Spiking Neural Networks. These are, as well, attempts at mimicking the human brain's behaviour, in hopes to transfer some of its extreme efficiency to our computing capabilities.
Unlike traditional cameras, which capture images at a fixed frame rate, event-based cameras operate based on the principles of pixel-level temporal contrast. Each pixel in an event-based camera independently monitors changes in intensity and generates an event when a significant variation occurs. These events, also known as "spikes", contain information about the location, timing, and polarity of the intensity change.

2.1 Architecture and operation

The first attempts at designing retinomorphic vision devices were mainly aimed at demonstrating neurobological models and theories, lacking the attention at practical applicability. Many pixel designs that were interesting from a conceptual point of view could not yield practically usable devices because of limitations such as circuit complexity, low fill factors, or high noise levels. In more recent times the interest in transferring these biological models to industrial applications increased, and with some good results.
One of the biggest challenges faced while working on this objective is reproducing the "wiring" complexity of nature: the about one million retinal neurons that compose the optic nerve with their axons should find an analogous in a wire for each single pixel transferring data out of the array. While this is clearly not feasible for industrial-grade sensors, a possible solution, and thus a crucial step towards practical applications, is represented by *Address-Event Representation (AER).*
AER is a communication and encoding scheme that enables the efficient transmission and processing of spikes, taking advantage of the difference in bandwidth between a neuron and a typical digital bus. This allows designers to multiplex the signals from the neurons over a few wires, making use of a large number of switches instead. The low-frequency (10-1000 Hz) spikes are encoded with an address (e.g. the x and y coordinates of the pixel that generated them within the array), while the timestamp is encoded implicitly and can be made explicit when tranferring data to a non-AER

processor.

One first example of event-based vision sensor is represented by the *DVS (Dynamic Vision Sensor)*. It was first proposed by Lichtsteiner and Delbruck in 2005 [16, 1] and is built on a simplified model of the retina composed of 3 layers: a photoreceptor circuit, a differecing circuit (corresponding to the bipolar cells of the retina) and comparators (corresponding to the ON/OFF ganglion cells). An abstracted pixel schematic is visible in figure 2.1.

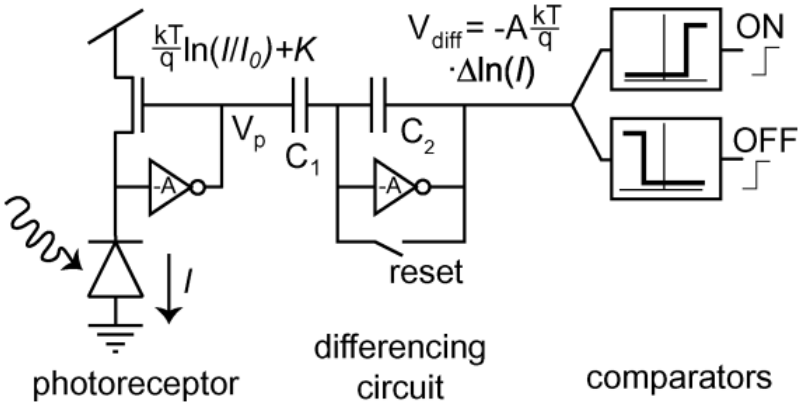


Figure 2.1: DVS pixel schematic [1]

The first stage transduces photocurrent to a voltage proportional to the logarithm of light intensity

$$V_p = V_{DC} + A_\nu U_T \ln(I)$$

The second stage amplifies V_p by C_1/C_2. Every time this amplified voltage changes over a fixed threshold V_{th}, the charge integrated at C_2 is reset. As a result, the pixel generates a spike every time the incident light varies by $\ln I(t_2) - \ln I(t_1) = \pm\theta_{ev}$. The third stage uses comparators to assign a polarity to the event generated. Each pixel is decoupled from the others, meaning it responds asinchronously to changes in the local intensity with a time resolution in the order of microseconds. The output is thus dependent of the scene dynamics, and in particular on the reflectance changes. The traditional idea of frame is completely surpassed. An example of DVS pixel operation is shown in figure 2.2

DVS pixels do not acquire any information on the scene absolute light intensity. The contrast sensitivity, i.e. the minimum variation that can be detected by a pixel is

$$\theta_{ev} = \left| \ln \frac{I(t_2)}{I(t_1)} \right| \approx \left| \frac{\Delta I}{I} \right|$$

In practical applications, values of contrast sensitivity of around 1% have been reached.
Different improvements to the DVS pixel design have been proposed since its first description, mainly with the aim of making it more compact.

Though in many applications DVS events are sufficient, some still require some form of static output.
This idea of combining the event-based output with information on the absolute

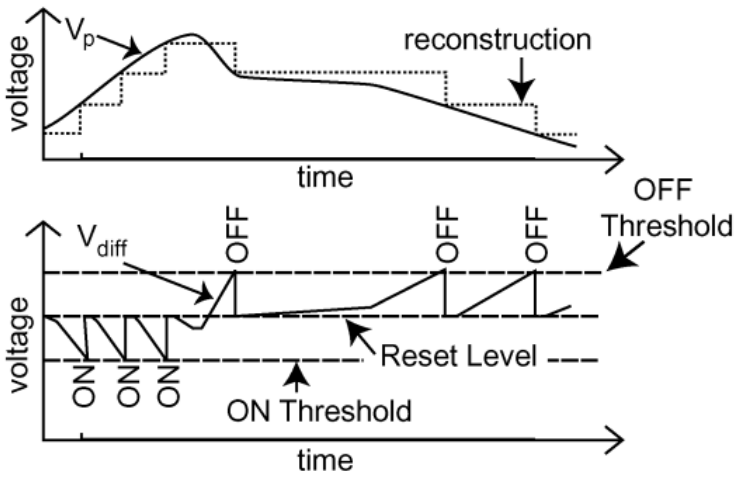


Figure 2.2: Principle of operation of a DVS pixel [1]

value of light intensity is at the basis of the *ATIS (Asyncronous Time-based Image Sensor)* pixel design [17]. This idea is not new, as one of the major drawbacks of traditional cameras is redundancy: ideally, only information that is unknown should be acquired and processed. Different solutions have been proposed, such as frame differencing, but they still rely on frame acquisition and their temporal resolution is limited to the frame rate achievable by the camera.
ATIS introduces a conditional exposure measurement circuit, combined with a DVS change detector, in order to transfer this compression process to the pixel level. The DVS part asynchronously triggers the acquisition of the grayscale value only if an intensity change greater than a certain threshold is detected in that pixel. A functional diagram of the ATIS pixel is represented in figure 2.3. The disadvantages of the ATIS are a large area of the pixel (at least double than that of the DVS pixel) and the limitations in dark environments: the time between two intensity readouts can be long and new events may interrupt it [2].

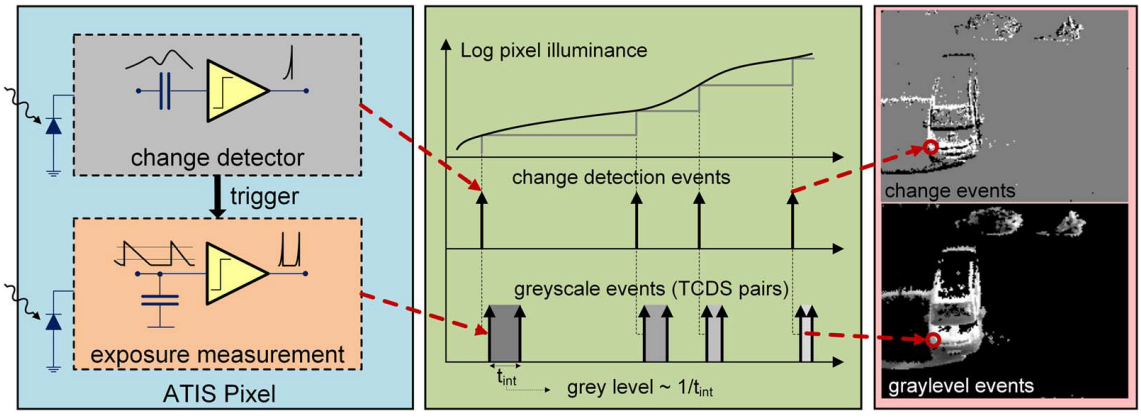


Figure 2.3: ATIS pixel functional diagram [2].

A different approach is represented by *DAVIS (Dynamic and Active pixel Vision Sensor)* [18]. This design merges the DVS pixel with gray-level information on the scene acquired by a traditional CMOS pixel. The schematics is represented in figure 2.4. Since the photodiode is shared between the two, the area of the DAVIS pixel is just 5% larger than the DVS, needed to accommodate the extra transistors.
This allows to bring back the "familiar" frames, with all the research that has been

conducted on them, and with their already discussed limitations as well. The role of the DAVIS could be to bring together these two paradigms, for example triggering frame capture efficiently by analysing the scene dynamics.
In this work a DAVIS sensor was used: the synchronous frames were used to extract machine learned features from the event stream.

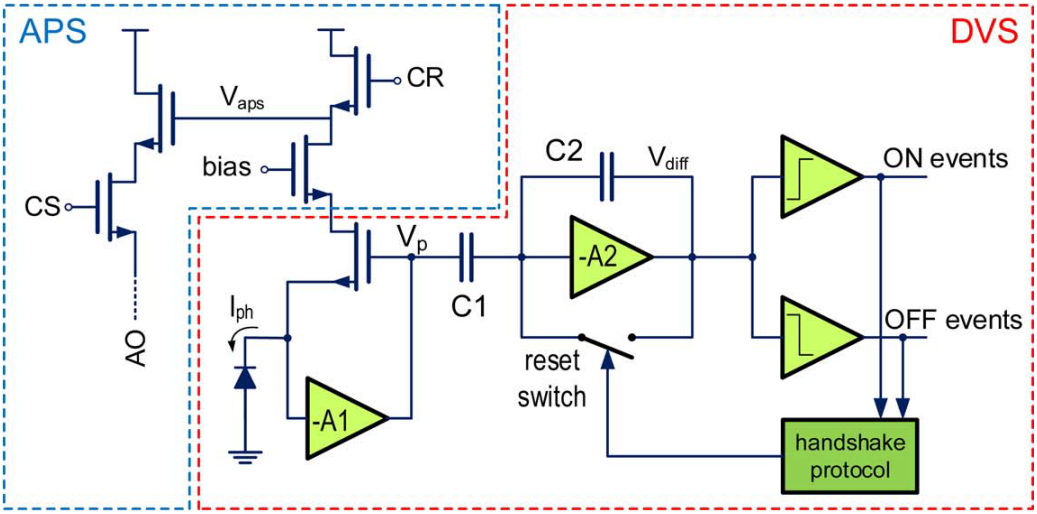


Figure 2.4: DAVIS pixel schematics [2].

2.2 Advantages of event-based cameras

As briefly mentioned above, event-cameras offer many advantages over traditional cameras:

- High Temporal Resolution: Event-based cameras offer extremely high temporal resolution, capturing changes in the scene with microsecond precision. This enables the detection and tracking of fast-moving objects that might be missed or blurred by traditional cameras, making them suitable for applications such as robotics, autonomous vehicles, and sports analysis.

- Low Latency: The asynchronous nature of event-based cameras allows them to respond almost instantaneously to changes in the scene, reaching sub-millisecond latency in real-world applications [4]. This low latency is crucial in real-time applications where immediate feedback or action is required.

- High Dynamic Range: Since each pixel operates independently and in logarithmic scale, event-cameras can reach dynamic ranges better than $120dB$, when high-quality frame-based cameras only reach about $60dB$.

- Low Power Consumption and Data Throughput: Event-based cameras consume significantly less power compared to traditional frame-based cameras. Since they only transmit data when an event occurs, ignoring redundant data, event-based cameras are highly energy-efficient, making them suitable for battery-powered or resource-constrained devices.

- Robustness to Motion Blur: Traditional frame-based cameras capture images over a finite exposure time, resulting in motion blur when objects move rapidly.

Event-based cameras, with their high temporal resolution, capture only the changes in the scene, eliminating motion blur and ensuring sharper images, even in dynamic environments.

An illustrative example of some of these features is shown in figure 2.5. These characteristics make them a potentially disruptive technology in applications such as:

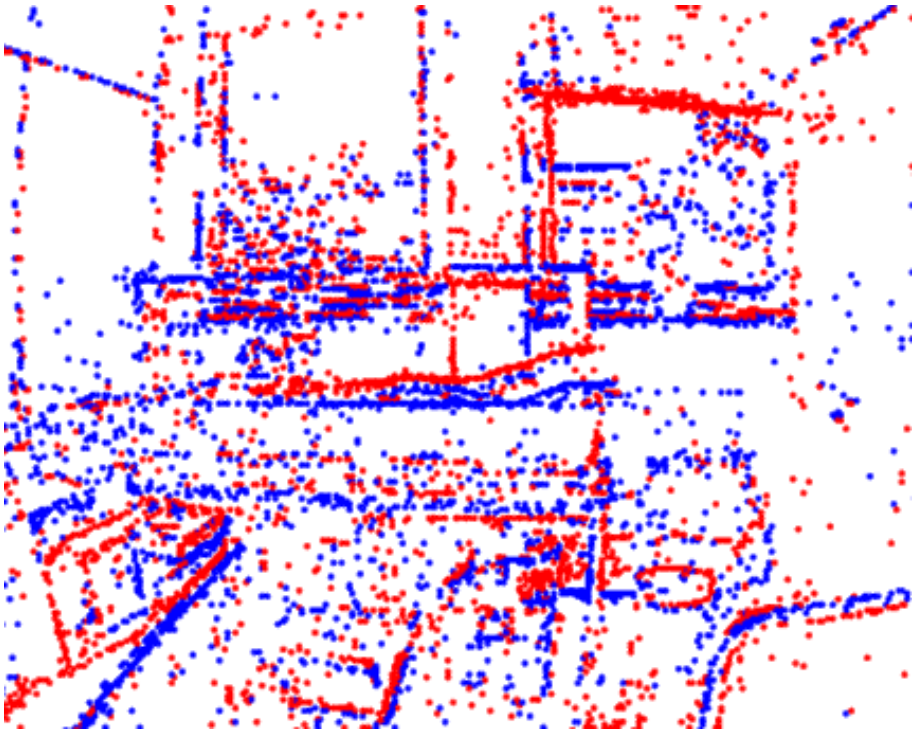
- Robotics and Autonomous Systems: Event-based cameras have transformative potential in robotics and autonomous systems. Their low latency and high temporal resolution make them ideal for visual feedback in control systems, enabling robots to react quickly to their surroundings. They can enhance object detection, tracking, and pose estimation, contributing to improved perception and manipulation capabilities.

- Surveillance and Security: Event-based cameras offer significant advantages in surveillance and security applications. Their ability to capture fast-moving objects and operate in challenging lighting conditions makes them effective in tracking and analyzing events in real-time. They can aid in detecting anomalies, recognizing objects, and monitoring crowded environments with enhanced accuracy and efficiency.

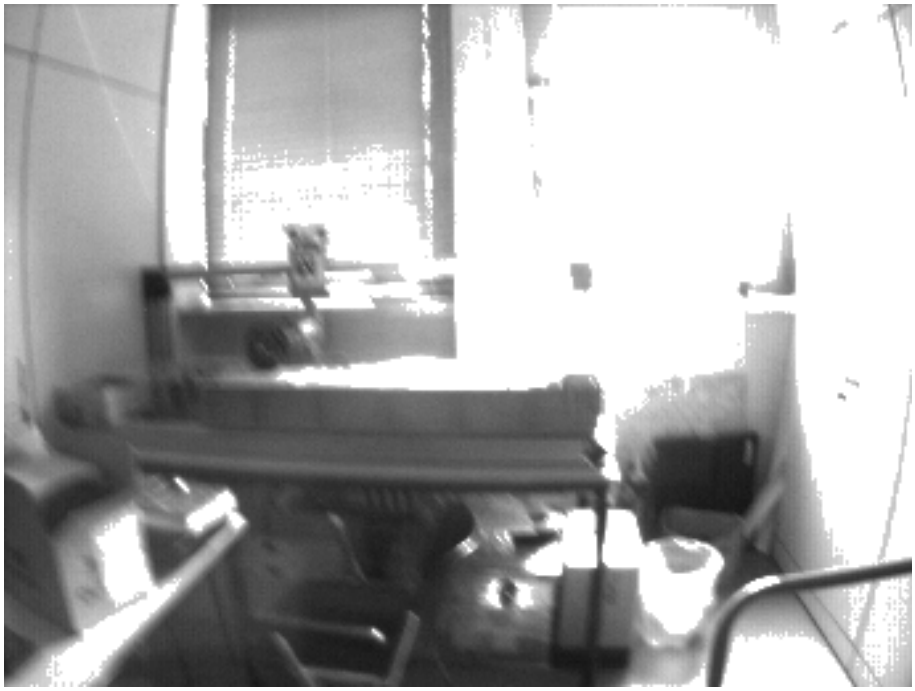
- Augmented and Virtual Reality: Event-based cameras can enhance the user experience in augmented and virtual reality (AR/VR) applications. Their low latency and high temporal resolution enable faster tracking of user movements and gestures, resulting in more immersive and responsive AR/VR experiences. Moreover, event-based cameras can reduce the computational load by transmitting only the relevant visual changes, leading to more efficient processing in AR/VR systems.

- Automotive and Transportation: Event-based cameras have the potential to revolutionize the automotive and transportation industries. Their high-speed and low-latency capabilities make them suitable for advanced driver-assistance systems (ADAS) and autonomous vehicles. Event-based cameras can improve object detection, pedestrian tracking, and collision avoidance in real-time, enhancing road safety.

- Space: Event cameras have emerged as a promising technology for space applications, offering unique advantages over traditional frame-based cameras. They have the potential to revolutionize areas such as visual odometry, navigation, and object tracking in space missions. With their low-latency motion estimation capabilities, event cameras can enhance spacecraft navigation, enabling precise trajectory tracking and facilitating autonomous operations. Furthermore, their ability to capture fast and dynamic events makes them well-suited for object tracking, collision avoidance, and situational awareness in space. Additionally, event cameras can contribute to spaceborne imaging, capturing high-speed phenomena with their high dynamic range and low power consumption. They can also play a vital role in robot-assisted space operations, improving the accuracy and efficiency of tasks such as maintenance, repairs, and sample collection. The sparse and efficient data provided by event cameras can be utilized for on-board image compression, reducing data transmission and storage requirements while preserving critical information.



(a) Event frame generated with N=5000 events. Different colors indicate different polarities.



(b) Grayscale frame generated by the same view.

Figure 2.5: Example of acquisition with an event camera. The event camera is able to detect the building and tree branches outside the window, while the traditional sensor is saturated. Motion blur is also not present in the event frame. Data acquired with a DAVIS 346 [3].

Lastly, event cameras have the potential to enhance space surveillance and situational awareness by continuously monitoring a wide field of view, aiding in space traffic management and the monitoring of space debris populations. As event camera technology continues to advance, their integration into space missions is expected to bring about significant advancements in various aspects of space exploration and satellite operations [19, 20, 21].

2.3 Event processing

In the previous sections the potential of event-cameras was discussed, but the new paradigm also poses many challenges, mainly in the way we can process this new type of data and extract useful information from it. Vision algorithms developed for frame-based cameras can't be directly applied to event streams: for once, the output from event-cameras is sparse, whereas traditional frames are dense, but most importantly event streams encode different photometric properties and they depend on the current and past relative motion between the camera and the scene.
Another challenge that was only tangentially touched above is noise. While some noise is inherently present in every sensor, in this case it is strictly related to the contrast sensitivity: setting it too low results in a lot of events being generated. For this reason, while lower values have been achieved in a laboratory setting, typical DVS have a threshold between 10% and 50% intensity change [4]. A complete model of the noise that affects the sensor in various illumination condition has not yet been achieved, but some noise filtering techniques have been developed, mainly based on the idea that "real" events should be more correlated than noise events. Some of these include:

- Refractory Period Filtering: One common noise filtering technique is to employ a refractory period, where events are ignored within a short time window after a previous event occurs. This filtering approach helps eliminate duplicate events caused by noise.

- Temporal Consistency Filtering: This technique involves examining the temporal consistency of events within a spatiotemporal neighborhood. By considering the consistency of event timestamps and spatial locations, events that do not adhere to the expected temporal patterns can be identified and filtered out as noise.

- Spatial Consistency Filtering: Similar to temporal consistency filtering, spatial consistency filtering considers the spatial relationships between events. Events that occur in spatially disconnected regions or deviate significantly from the expected spatial patterns can be considered noise and filtered out.

- Adaptive Thresholding: Adaptive thresholding methods dynamically adjust the threshold for event detection based on the local contrast characteristics of the scene. By adaptively setting the threshold, noise events with low contrast can be suppressed while preserving important events with higher contrast.

- Machine Learning-based Approaches: Various machine learning techniques can be employed for noise filtering in event cameras. These approaches can learn to distinguish noise events from genuine events by training on labeled event

data. Machine learning models can capture complex spatiotemporal patterns in the event stream and provide effective noise filtering.

2.3.1 Event representation

One first useful step in the development of algorithms for processing event streams is exploring different possible representations of the data. This is helpful because one single event does not convey enough information and thus aggregating them in some way becomes a necessity.

- Individual Events: Each event is considered individually in the form $e_k = (\mathbf{x}_k, t_k, p_k)$, where $\mathbf{x}_k$ is a vector representing its location on the image plane, t_k is its timestamp and p_k represents its polarity, i.e. the sign of the intensity variation. Figure 2.6.

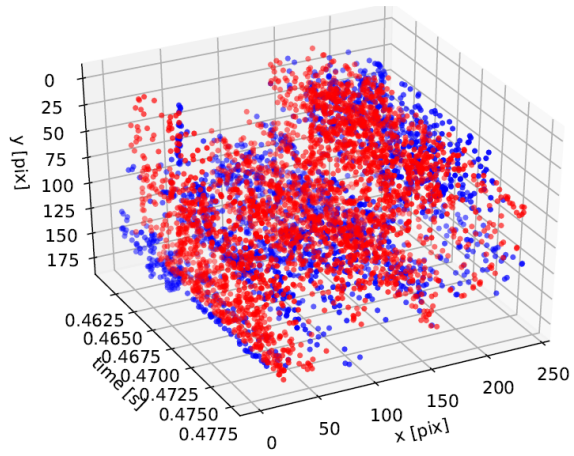


Figure 2.6: Example of individual events representation [4].

- Event Packets: Events are aggregated based on their temporal proximity and processed together: $\varepsilon = \{e_k\}_{k=1}^{N_e}$. The choice of the number of events that compose each packet becomes critical and highly dependent on the processing algorithm to be used.

- Event Frames: Event packets are processed in a simple way, for example accumulating polarity pixel-wise, to obtain a frame-like object. While this allows to use the many algorithms developed for frame-based vision, the risk is to not fully take advantage of the peculiarities of event-based vision, like asinchronicity and low latency. Many feature detection algorithms still make use of some form of event frames because of their simplicity and ease of interpretation and because they convey information on the edges present in the scene, which are the most informative region of the image. Figure 2.7.

- Time Surfaces: Each pixel stores a single time value, typically the timestamp of the last event at that pixel, composing a "frame" that contains dynamic information on the scene motion. While time surfaces can achieve a high compression of the information, their limits emerge when facing a highly textured scene, in which the frequent spiking of the pixels can lead to the loss of information. Figure 2.8.

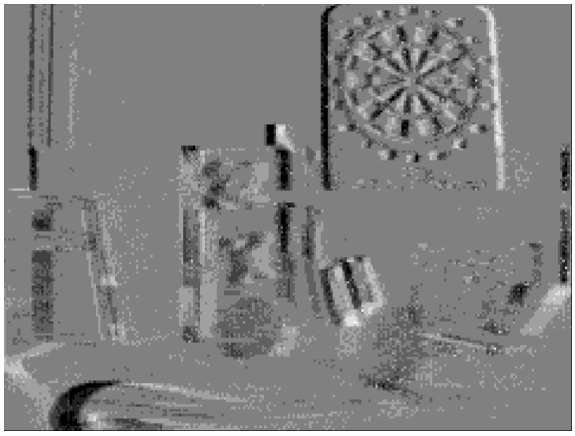


Figure 2.7: Example of event frame representation [4].

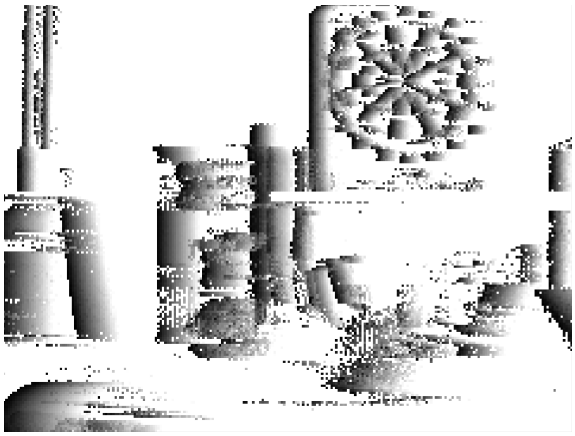


Figure 2.8: Example of time surface representation [4].

- Voxel Grids: Events are represented in a 3D histogram, where each voxel corresponds to a particular location and time interval. Figure 2.9.
- Motion Compensated Images: Based on the hypothesis that events are triggered on the pixels traversed by an edge moving on the image plane. Maximizing the alignment of the events by warping them, the motion of the edge can be estimated. Figure 2.10.

2.3.2 Processing algorithms

Different approaches have been proposed to extract information from event streams. The choice of the processing algorithm is highly linked to the representation used and to the hardware available (dense representations are the natural choice for traditional processors, while event-by-event representations make the most sense when coupled with neuromorphic hardware) [4].
One main classification that can be made distinguishes between methods that operate on an event-by-event basis, meaning that the state of the system can change upon the detection of one single new event, and methods that operate on a number of accumulated events, thus introducing some latency.
This last category is largely composed of methods that preprocess the events to be analysed in a representation that allows the application of algorithms developed for frame-based computer vision. While this, on the one hand, means working with algorithms that have been studied and used for a long time, on the other presents

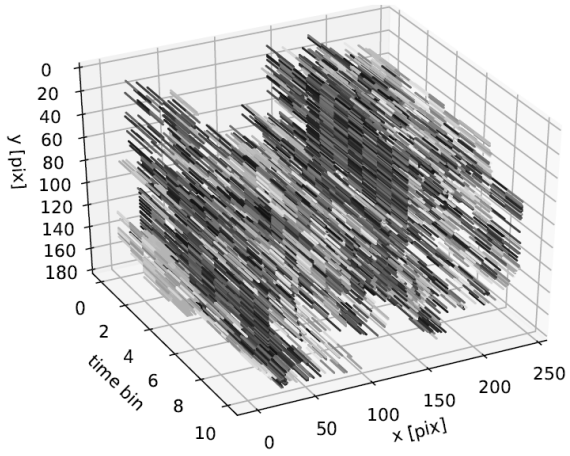


Figure 2.9: Example of voxel grid representation [4].

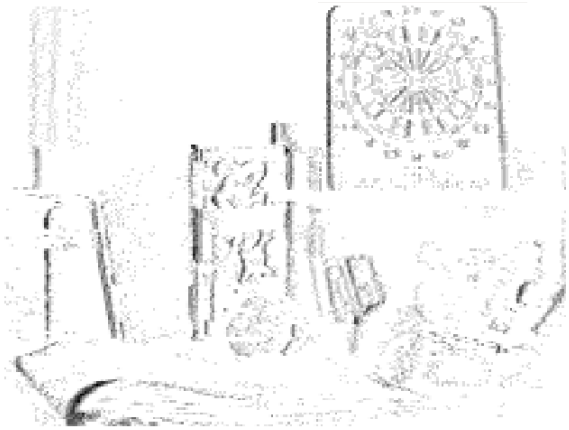


Figure 2.10: Example of motion compensated image representation [4].

the risk of not fully exploiting the advantages offered by event-based vision.

eHarris [5] is an adaptation of the well-known Harris corner detector, that identifies a corner if a matrix containing the first derivatives of intensity has two large eigenvalues, i.e. if the intensity gradients centered around that pixel occur in two different directions. The authors use a time surface composed of a fixed number of events, in order to avoid the dependency on the scene speed. One time surface is generated for each polarity and they are then binarized depending on the presence of an event, as shown in figure 2.11.

The surface obtained is thus defined as

$$\Sigma_b : \mathbb{N}^2 \rightarrow \mathbb{N}$$

$$\Sigma_b(\mathbf{p}) = \begin{cases} 1 : \exists\ \mathbf{e}\ at\ \mathbf{p} \\ 0 : \nexists\ \mathbf{e}\ at\ \mathbf{p} \end{cases}$$

where $\mathbf{e} = (\mathbf{p}, t, pol)$ represents each event and $\mathbf{p} = (x, y)^T$ is its position.
The gradients of Σ_b are then computed, and, after filtering them with a Gaussian window function, a score is assigned to each event based on the previously mentioned eigenvalues. If this score is greater than a set threshold, the event is labelled as a corner event.

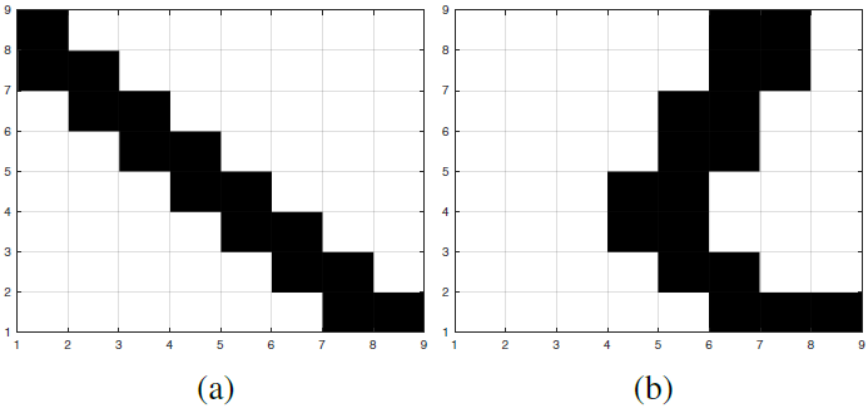


Figure 2.11: Examples of the time surface used in eHarris for an edge (a) and a corner (b). Its gradients are computed and, following the application of a Gaussian window for smoothing, the Harris score is calculated to determine whether the analyzed event is a corner [5].

Another classical algorithm that has been reproduced is FAST (Features from Accelerated Segment Test) [22]. The idea it is based upon is checking a 16-pixel circumference around the candidate corner pixel. If a sufficient number of adjacent pixels are found to be brighter or darker than the candidate pixel by at least a threshold value, the candidate is considered a potential corner. The event representation chosen in this case [6] is the Surface of Active Events (SAE) described in [23], which is a simple time surface defined by

$$SAE : (x, y) \rightarrow t$$

The events are separated by polarity and treated independently in this case, too. A FAST-like algorithm is applied to the position of the current event (instead of iterating over all the pixels, as it is done in a traditional frame): since the current event is always the latest, meaning it always represents the maximum value of the SAE, comparing its value to the surrounding pixels does not provide any information. Some modifications to FAST are thus necessary.
The idea is that a moving corner creates two clearly separated regions, one containing recent events and one containing older events. Identifying contiguous pixel with a higher value than the surrounding ones will thus correspond to finding corners. A circular search (similar to FAST) allows the algorithm to be isotropic and efficient. As can be seen in figure 2.12, two circles around the latest event are considered for higher accuracy. Since the SAE is updated with each new event arrival, the algorithm maintains some of the asynchronicity characteristic of event-based vision. The performance of *eFAST* is slightly worse than eHarris, but it allows for a significantly lighter computational cost, as it avoids complex computations, such as derivatives.

One of the limitations of *eFAST* is its inability to detect corners when the scene dynamics causes the SAE to form circular arcs of newest events greater than 180 deg, as depicted in figure 2.13. This is necessary to avoid identifying straight edges as corners.

From this shortcoming springs *Arc* [7, 24]. One first difference with *eFAST* is the SAE used: in *Arc* an additional variable is considered, the refernce time t_r. The new SAE is thus defined as

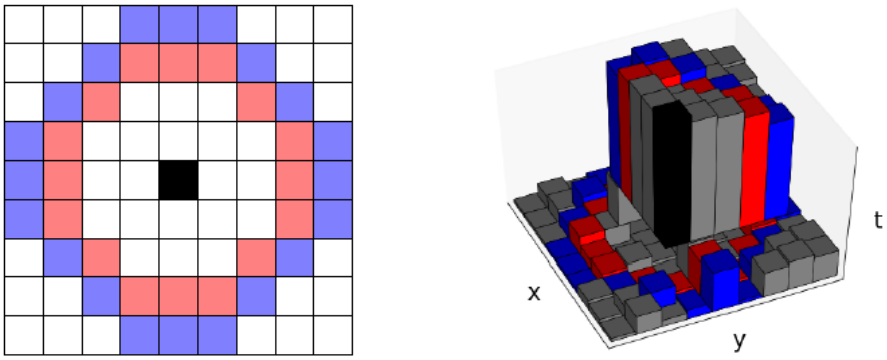


Figure 2.12: Representation of the *eFAST* algorithm pattern [6].

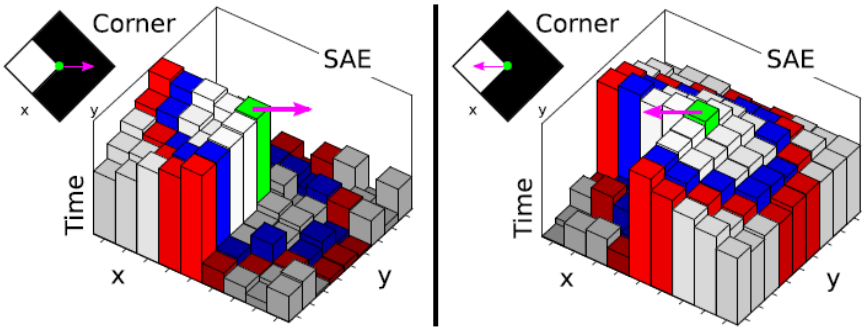


Figure 2.13: The same corner, under different motion directions, can generate two very different SAEs. In the right case *eFAST* is not able to detect the corner [7].

$$SAE^* : (x, y) \rightarrow (t_r, t_l)$$

When a new event is detected at time t, the latest timestamp is always updated $t_l \leftarrow t$, while the reference timestamp is only updated if the previous event in the same location does not fall into a time-window k, such that $t_r \leftarrow t$ if $t > t_l + k$. When querying for values in a specific location, the reference time is returned and the algorithm only considers the events that update t_r. This new SAE is inspired by the observation that, because of hardware limitations, there exists a minimum time between two consecutive events. An example is depicted in figure 2.14.

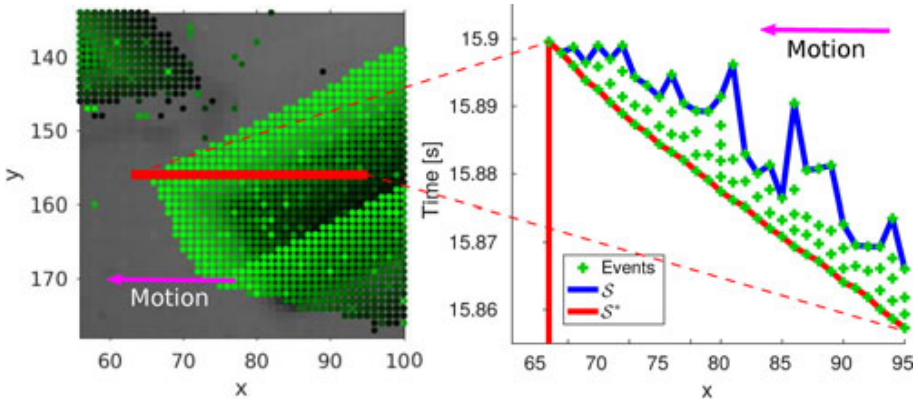


Figure 2.14: Example of the SAE proposed in [7] for the *Arc* algorithm [7].

The algorithm is then similar to *eFAST*: a circular set of pixels around the latest

event is considered. Among these, the one with the most recent event and the two adjacent ones (E_{CCW}, E_{CW}) are considered. This arc is then expanded in the direction of the higher SAE* value between E_{CCW} and E_{CW} until a pixel where SAE* is lower than the other is found (example in figure 2.15). The pixel is marked as a corner if this arc or its complementary has a length included in a fixed window. In [7, 24] this detector is coupled with a dedicated tracking algorithm that takes into account different hypotheses using a tree graph.
Experimental results show that *Arc* performs similarly to *eHarris*, but with a significantly higher False Positive Rate. On the other hand its very limited computational cost allows it to run in real time.

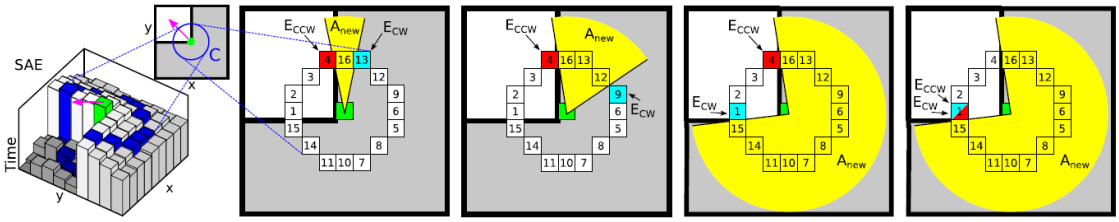


Figure 2.15: Representation of the *Arc* algorithm [7].

A different attempt at adapting the Harris corner detector to an event stream is represented by *luvHarris* [8]. The arc-based detectors described above have the great advantage of being fast to compute, and that's because they only consider a few pixels around the corner candidate. However this characteristic also causes them to be highly susceptible to noise in those pixels. To overcome this problem *luvHarris* makes use of the robust Harris algorithm, but with an architecture that allows it to run in real time.
The proposed method is composed of two parts: a threshold-ordinal surface (TOS) that is updated each time a new event is recorded and a Harris-score computation that takes place any time the previous instance is finished, independently of the number of new events detected.
The TOS is defined as

$$TOS : (x, y) \rightarrow 0 \cup [(255 - T_{TOS}), 255]$$

where T_{TOS} is a threshold value. The advantage of this time surface are that it has an unused range between 0 and $255 - T_{TOS}$, increasing the signal-to-noise ratio of gradients for the Harris algorithm and that it is forced to return to 0 after an edge has passed (figure 2.16).

The Harris score is computed over the most up-to-date TOS as frequently as the computation allows. The output of the Harris detector constitutes a look-up table L, which is used for multiple events (the number varies depending on the event throughput): if $L > T_R$ the event is marked as a corner.
This two-part algorithm allows to modulate computations depending on the scene structure and dynamics, granting more flexibility and increasing the event frequency that can be handled in real time. Since the two parts are completely decoupled, it also lends itself well to parallel operations.
While this architecture seems to work well in high-rate event streams, it becomes less efficient than other algorithms when the event-rate decreases, as the whole Harris score computation becomes redundant in this scenario.

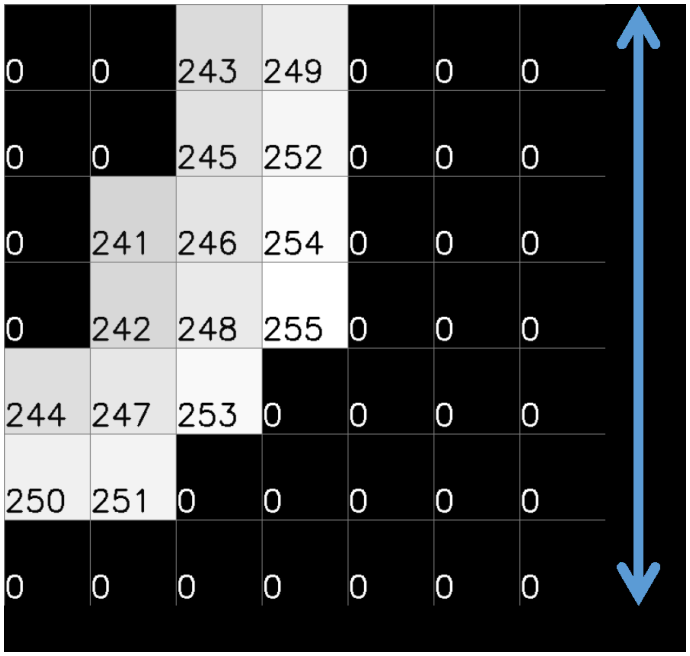


Figure 2.16: Example of the threshold-ordinal surface described in [8].

Other than these attempts at adapting algorithms designed for frame-based vision, some corner detectors have also been designed specifically with the event-based paradigm in mind.

HASTE [25, 9] is a detection and tracking algorithm that treats the problem as an optimization problem, with the aim of matching the current view to a feature template. The high temporal resolution and asynchronicity of the event stream are what make this approach possible.
Each feature is represented by an $n \times n$ template T and a vector $\mathbf{x} = \{x, y, \theta\}$ containing its position and orientation in the image plane. Assuming the optimal feature state at each instant can be determined by a window ε containing the latest m events registered in the feature patch, upon arrival of a new event its coordinates $\mathbf{p}$ are transformed to the feature's reference frame following $\mathbf{x}$, such that $\mathbf{p'} = R^T(\theta)(\mathbf{p} - [x, y]^T)$. Each time a new event is registered in the patch, it replaces the oldest event in the window and the template is also updated.
The optimization problem

$$\mathbf{x}^{(k+1)} = \arg\min_{\mathbf{x} \in X} f(\varepsilon^{(k+1)}, T^{(k+1)}, \mathbf{x})$$

is approximated over a discretized space of solutions H

$$\mathbf{x}^{(k+1)} = \arg\min_{\mathbf{x} \in H(\mathbf{x}^{(k)})} f(\varepsilon^{(k+1)}, T^{(k+1)}, \mathbf{x})$$

The space of solutions considered is defined as

$$H(\mathbf{x}) = \{(\mathbf{x} = \{x, y, \theta\}) \cup \{x \pm \Delta x, y \pm \Delta y, \theta \pm \Delta\theta\}$$

where the values considered are $(\Delta x = \Delta y = \pm 1, \delta\theta = 0°)$ and $(\Delta x = \Delta y = 0, \delta\theta = \pm 4°)$.

Upon arrival of a new event the correlation score for each hypothesis is computed and when one of this scores is better than that of the null hypothesis (which is the previous optimum), it becomes a new optimal state (figure 2.17). A new set of hypotheses is then computed and the process is repeated, meaning the feature is tracked in an event-driven fashion. Different variants of the score function f have been proposed and evaluated by the authors in [9].

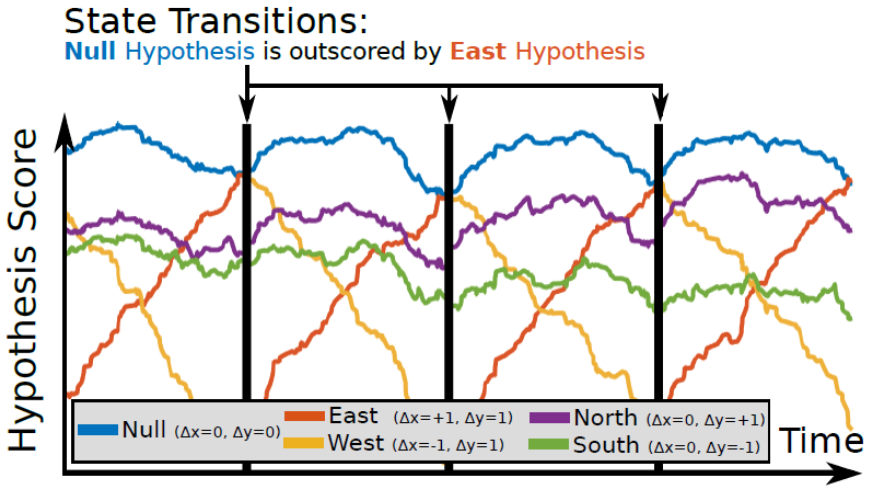


Figure 2.17: For each incoming event, the state is approximated with the best scoring hypothesis (in blue). In this example (using only 5 hypotheses), the feature is moving in the $+x$ direction [9].

eCDT [10] is a feature detection and tracking algorithm that makes use of clustering in an attempt to avoid frame-based representations.
The first step it includes is separating clusters of events with the same polarity observing that they are bounded by clusters of different polarity, as illustrated in figure 2.18. The authors gave this algorithm the name *KCSCAN*.

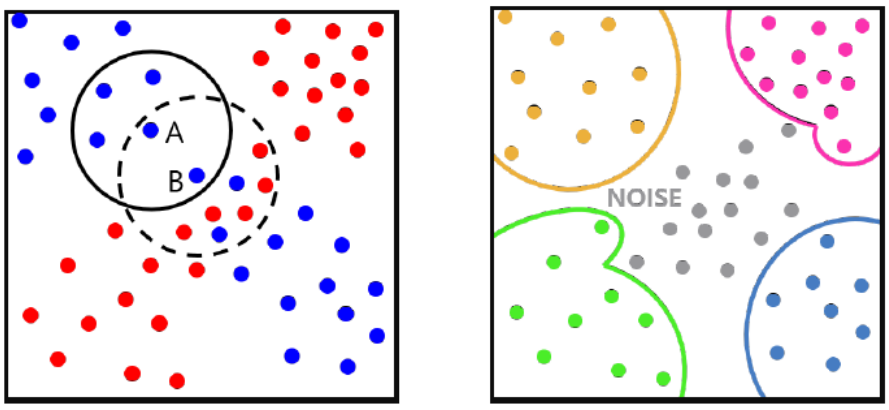


Figure 2.18: Simplified 2D representation of *KCSCAN*. Point A is a core point because it is surrounded by points of equal polarity, while point B, being surrounded by points of different polarity is classified as noise [10].

Clustering equal polarities is shown to be weak in handling sudden polarity inversions caused by changes in edge direction. To address this, a cluster descriptor is created to enable continuous tracking even with inverted polarities. Data association between clusters is crucial, and the spatiotemporal neighboring characteristic

of events is exploited. By searching the proximity of space where a cluster ends, lost features can be recaptured. The search is limited by a temporal search time. The problem is formulated as finding matching descriptors in a spatiotemporal space near the termination of a cluster. In situations of sudden movement where polarity inverts, the shape created by the edge remains constant in a short time window. Therefore, the problem only requires matching the shape and position of a terminating cluster with newly created adjacent ones. Cluster head descriptors are used to find preceding clusters from the same edge, and cluster tail descriptors are used to find future clusters. A mutual information-based descriptor matching method is incorporated, where the tail descriptor of one cluster is matched with the head descriptor of another cluster. The sets of tail and head pixel locations are collected for a short time interval. The mutual information (intersection over union) is computed to measure the similarity between the tail and head descriptors. Clusters with the largest mutual information are grouped together, allowing for longer tracks. This approach enhances tracking by addressing polarity inversions and changes in direction, enabling more robust and continuous feature tracking, as shown in figure 2.19. The feature tracks are reconstructed by computing the center of gravity of events in each cluster in a temporal window.

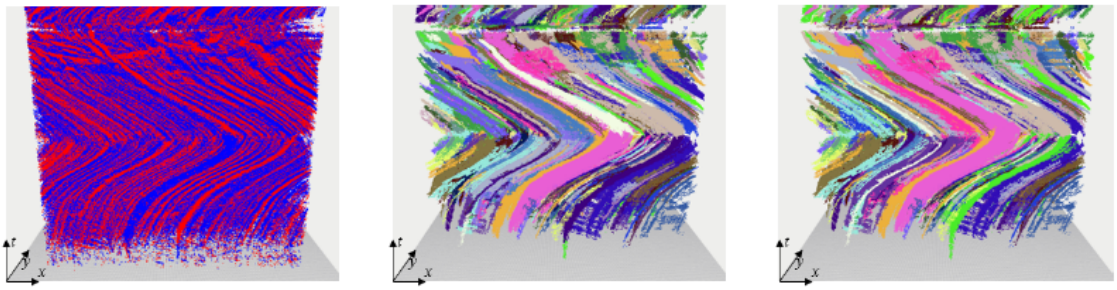


Figure 2.19: Raw events are clustered by polarity. Consecutive tracks are then concatenated to provide longer and more robust tracks [10].

While the performance of *eCDT* is substantially on par with *HASTE*, the resulting tracks display some more noise. This is caused by the fact that the edge features are simplified as single centroid points.

A different class of methods is composed of learned methods, rather than handcrafted as the ones described until now.

In [11] the authors propose a random forest classifier to discriminate corner from non corner events. In order to do so, they chose a novel representation: a Speed Invariant Time Surface that normalizes the times of events based on their local speed. Instead of sorting local patches, which is computationally expensive, a single value is maintained for each pixel location and polarity. When a new event arrives, a large value is stored at its pixel location, and the values for surrounding locations are progressively reduced. This process ensures that the profile of the event wave remains consistent regardless of the contour's speed.
The Speed Invariant Time Surface algorithm initializes the surface to 0 and updates it for each incoming event. The neighboring pixel locations within a specified radius are considered, and if a neighboring value is larger than the current pixel's value, 1 is subtracted from it. Finally, the current pixel's value is set to the square of the radius. The algorithm ensures that the Speed Invariant Time Surface remains

within the range of 0 to $2r + 1$, where r is the specified radius.
The Speed Invariant Time Surface exhibits a consistent slope for edges moving at different speeds, while the other methods result in varying slopes (figure 2.20). It also enhances contrast after the edge by reducing values, leading to improved results compared to the standard Time Surface and sorting normalization method.

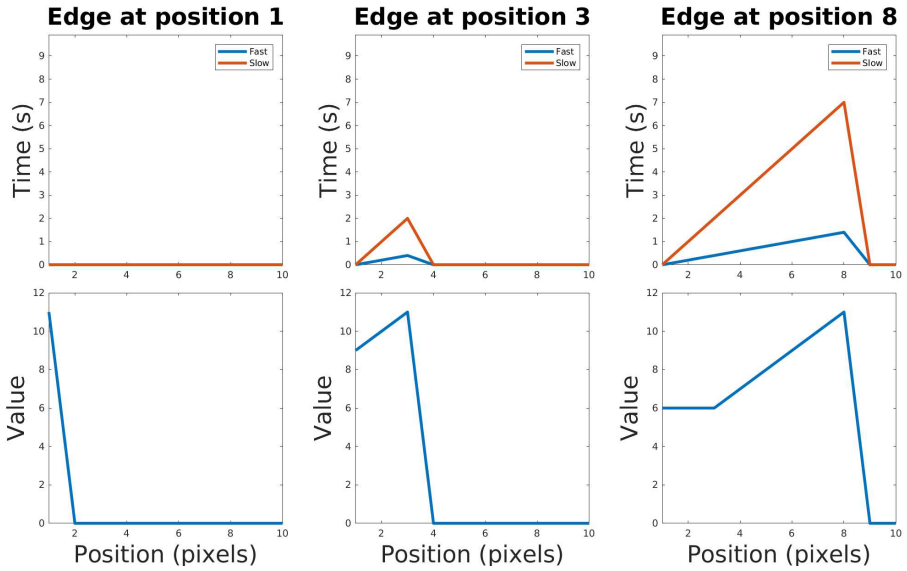


Figure 2.20: An edge moving from left to right at two different speeds creates the same slope in the Speed Invariant Time Surface (bottom), a different slope in the standard TS (top) [11].

A Random Forest classifier F consists of a collection of decision trees, F_l. The overall prediction of the Random Forest is obtained by averaging the predictions of each individual tree

$$F(s) = \frac{1}{L} \sum_{l=1}^{L} F_l(s)$$

Each decision tree recursively splits the input sample, $s = (s_1, ..., s_K)$, by branching its nodes until reaching a leaf node. At each node, a binary split function is applied to determine whether the sample should be sent to the left or right child node.
In the implementation presented in [11], the authors use decision stumps as the split functions, where a specific descriptor dimension, d_k, is compared to a threshold value, th. This choice is both computationally efficient and effective. The prediction of the decision tree is stored at the leaf node reached by the sample. In this case, this prediction represents the probability of the sample being a feature point. During the training phase, each tree is trained independently, following a greedy approach. The optimal split parameters, (d_k, th), are determined by minimizing the Gini impurity index through an exhaustive search. The dataset is recursively split into left and right child nodes based on these split parameters.
The dataset used for training was generated starting from the gray-level data provided by the ATIS sensor used, to which the Harris detector was applied in order to label the corner events as positive samples and non-corner events as negative samples.

A different approach is presented in [26, 12]. The inspiration is once again found in biology: the authors propose Vector Symbolic Architectures (VSAs), a family of models that encode the critical structures of cognition, namely variable binding, sequence, and hierarchy, into distributed representations. VSAs encode data structures holographically in a fixed-dimensional vector space. Operations such as addition, multiplication, and permutation are used to manipulate and build data structures in this vector space. To access or decode the components of a VSA-encoded data structure, the high-dimensional vector representing it needs to be decomposed into its primitive vectors. Though multiple approaches to this problem have been proposed, they resort to brute force methods or have limited data structure depth, resulting in limited applicability. The proposed solution is to factorize high-dimensional vectors into their primitives using a *resonator network* [27, 28], which is a type of recurrent neural network. The resonator network searches through the solution space using the VSA principle of superposition, avoiding an exhaustive search. It iteratively searches for potential factorizations until stable fixed points are found.
In particular, events are encoded with fractional power encoding (FPE) [29] as the sum of the exponentiated seed vectors $\mathbf{h}_0$ and $\mathbf{v}_0$ (for horizontal and vertical coordinates):

$$\mathbf{s} = \sum_{x,y \ in \ E} \mathbf{h}_0^x \otimes \mathbf{v}_0^y$$

where x, y are the coordinates of each event and $\otimes$ represents the Hadamard product. With this method event polarity is discarded and all the events are treated equally.
Figure 2.21 illustrates the division of the network into three components: (1) the hierarchical resonator responsible for tracking (represented in detail in figure 2.22), (2) the reference frame transformation that converts camera coordinates to map coordinates, and (3) the map update process. The hierarchical resonator takes the encoded image and the map as inputs and produces estimates of the transformation between them. The subsequent module utilizes these estimates to convert the encoded input image from camera coordinates to map coordinates. The transformed image in map coordinates is then utilized to update the map.

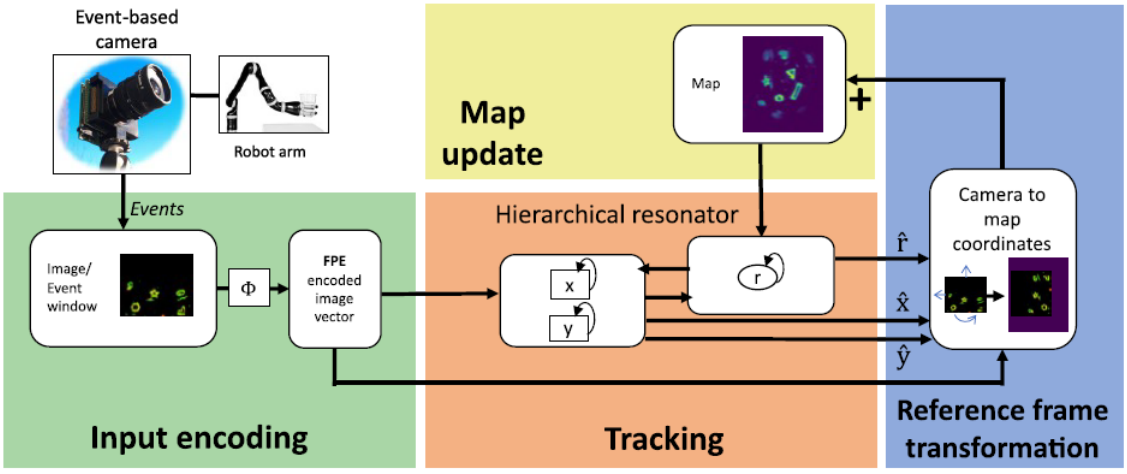


Figure 2.21: The architecture used for the visual odometry in [12].

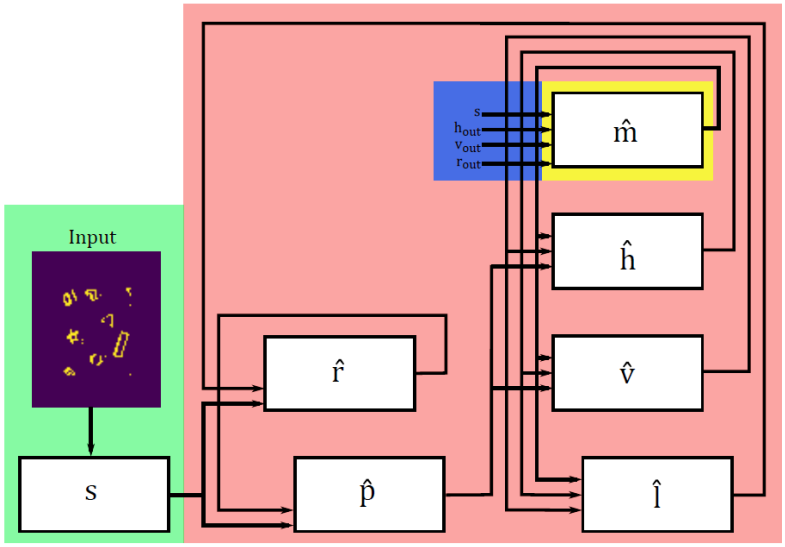


Figure 2.22: The resonator network described in [12]. **s** is the vector that encodes the events and $\hat{h}, \hat{v}, \hat{r}$ represent respectively the horizontal and vertical displacement and the rotation. $\hat{m}$ is the map [12].

[30] showcased the feasibility of reconstructing high-quality grayscale images from event streams. However, a drawback of this approach is that it requires a large neural network for accurate grayscale image reconstruction, rendering it unsuitable for low-power and real-time applications, which are typically the main motivations behind the utilization of event-based cameras. Several gray-level corner detectors rely on image derivatives rather than intensity values, and event cameras, sensitive to illumination changes, produce events directly linked to the spatio-temporal gradients of the scene. Therefore, the authors of [13] propose to train a recurrent neural network to predict image gradients from events instead of intensity values. The Harris detection algorithm is then applied using these gradients. Since the prediction of gradients from events is a simpler task compared to predicting images, a lightweight recurrent neural network is sufficient.
The architecture employed is a fully convolutional network with five layers, utilizing 3×3 kernels. Each layer consists of 12 channels and incorporates residual connections. The second and fourth layers adopt ConvLSTM, while the last layer, responsible for gradient prediction, is a conventional convolutional layer. For the remaining feed-forward layers, Squeeze-Excite (SE) connections are utilized. This architectural design ensures both efficiency and the ability to learn gradients from event data.
An immediate evolution of this is presented in [31]: the architecture used is the same, but this time the network is trained to output a sequence of heatmaps, whose local maxima correspond to the keypoint locations.

Method

This work is based on the detector presented in [31]. As mentioned above, the detector receives an event cube as input and outputs a sequence of heatmaps, whose local maxima correspond to the location of the keypoints.

3.1 Event cube representation

The use of event cubes as a useful representation of event streams was first proposed in [32]. Like the other representations, they still integrate the events generated in a time window to gather enough information and filter out noise, but, while in the other cases the events are compressed onto a 2D representation, in the event cube much more temporal resolution is maintained by adding a third dimension.
In particular the event cube is a $B \times H \times W$ tensor, H and W being the sensor height and width and B being the number of temporal bins in which the integration time is discretized. In this work, an event cube was produced every $N = 2000$ events, ensuring that the integration time (and thus the representation) still follows the scene dynamics. Each incoming event contributes to the two closest temporal bins of the event cube E through a trinagular kernel and based on its polarity p_i

$$E(x, y, t) = \sum_i p_i \max(0, 1 - |t - t_i^*|)$$

where t_i^* is the normalized timestamp of the event

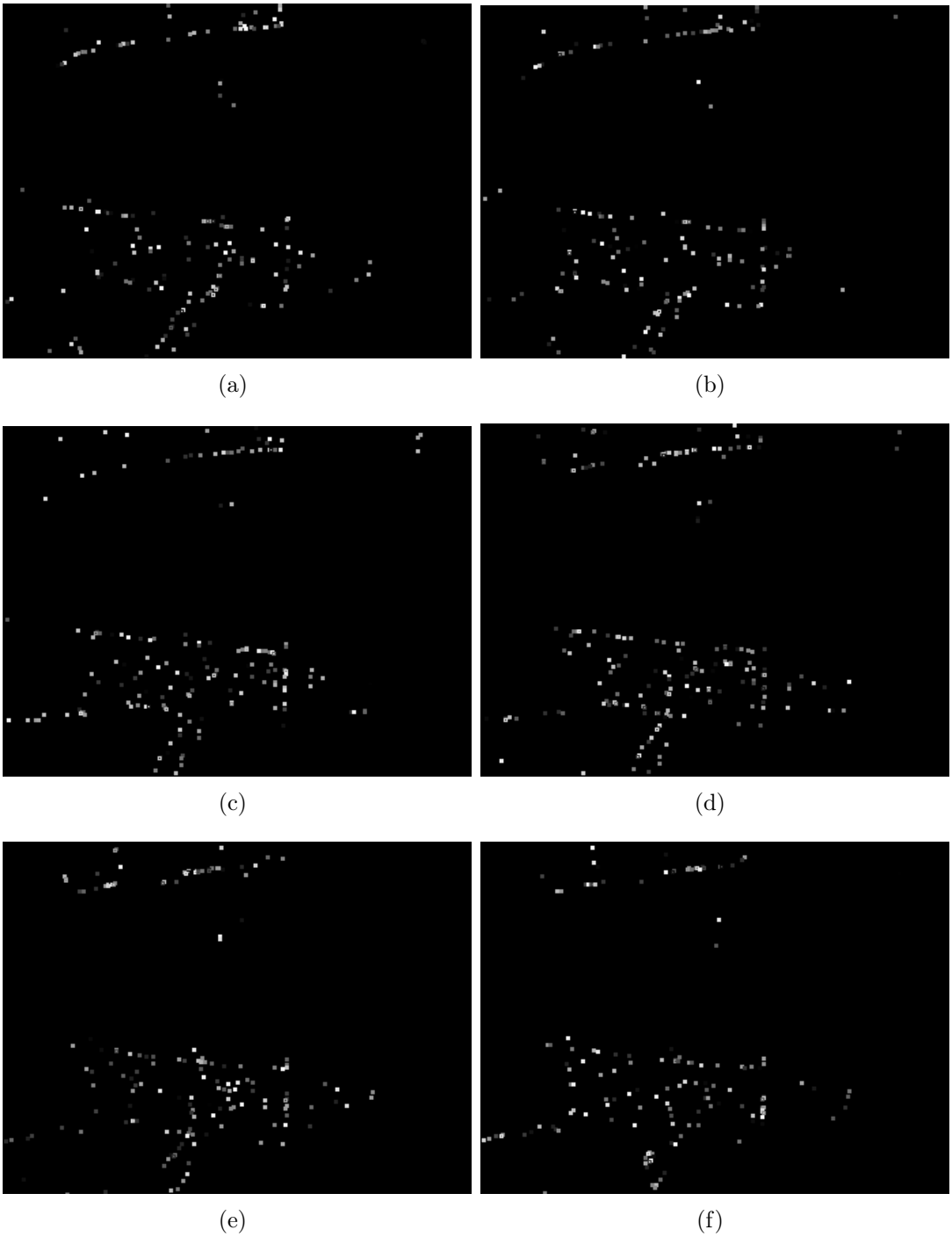
$$t_i^* = \frac{t_i - t_{min}}{\Delta T}(B - 1)$$

t_{min} being the beginning time of the integration period. For this work $B = 12$ was chosen. An example of event cube is presented in figure 3.1.

3.2 Architecture

A simple recurrent neural network is used to predict the heatmaps, similar to the one proposed in [33]. The event cube is fed into a Squeeze and Excitation block [34], followed by a Convolutional LSTM block [35]. These two layers are repeated and a final simple convolutional layer outputs the heatmaps. The values of the predicted heatmaps are constrained between 0 and 1 by applying the logistic function. Each layer has 12 channels and residual connections. The network is represented in figure 3.2.

A sequence of N_H heatmaps contitutes the output. Based on the results presented in [31], in this work $N_H = 12$ was chosen. This means that for each set of 2000 events, 12 time steps are considered, resulting in a very high temporal resolution



(a) (b)

(c) (d)

(e) (f)

Figure 3.1: Example of an event cube. The event frame generated by the same 2000 events is represented in figure 4.1c. (a) to (f) represent respectively temporal bins 1 to 6. Temporal bins 7 to 12 not shown.

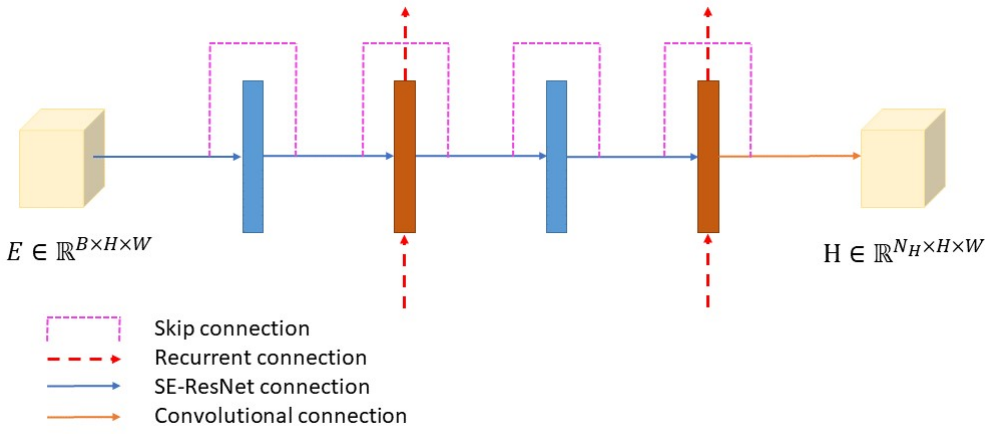


Figure 3.2: The architecture used in this work, first proposed in [13].

of the feature detection. At inference, the heatmaps undergo a local non-maximum suppression process for every patch of 7×7 pixels and every maximum with a value higher than $t_{keypoint} = 0.7$ is identifed as a keypoint and inserted in the corresponding "predicted frame".
To train the network a set of labelled locations $\hat{H}$, such that $\hat{H}(x, y) = 1$ if there is a keypoint at location (x, y) and $\hat{H}(x, y) = 0$ otherwise, was created. The loss function is then calculated as the binary cross-entropy between these labels and the heatmaps H predicted by the network

$$L = \sum_{h\in[1;N_H]} \sum_{(x,y)} BCE(H_h(x, y), a\hat{H}_h(x, y))$$

In [31] the necessity of hard negative mining during training was noted. It is caused by the fact that the number of keypoints is much smaller than the number of non-keypoints, which causes the network to converge to the solution of never detecting any keypoints, since this would already yield a relatively low loss value. In this work, the factor a was added, which only increases the weight of the keypoints' locations in the balance. $a = 500$ was empirically selected.

3.3 Training data

It is now clear that the training data should be composed by a $B \times H \times W$ event cube E and its respective $N_H \times H \times W$ set of labels $\hat{H}$.
Differently from [31], in which the network was trained on synthetic data generated by a simulator, in this work the training is done on data acquired with a DAVIS 346 event camera [3] by iniVation (figure 3.6). This device collects both the gray-level frame and the asynchronous events with a resolution of 346×260 pixels. It also includes an IMU, which however was not used for this work.
The labels are generated starting from the gray-level frames. First, they are analysed with a Harris detector to locate the corners. The corners in each two consecutive frames are matched by performing a nearest neighbor search. The matched corners are then filtered using a MSAC algorithm, which takes into account the epipolar constraint. Lastly, since event cubes have a much higher frequency than gray-level frames, the labels are obtained by linearly interpolating the positions of the filtered keypoints.

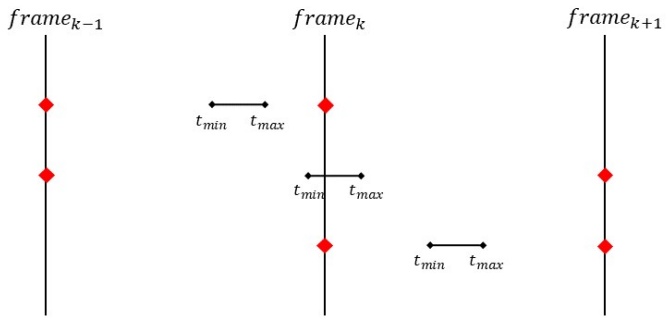


Figure 3.3: Selection of the frames used for interpolation in training data generation.

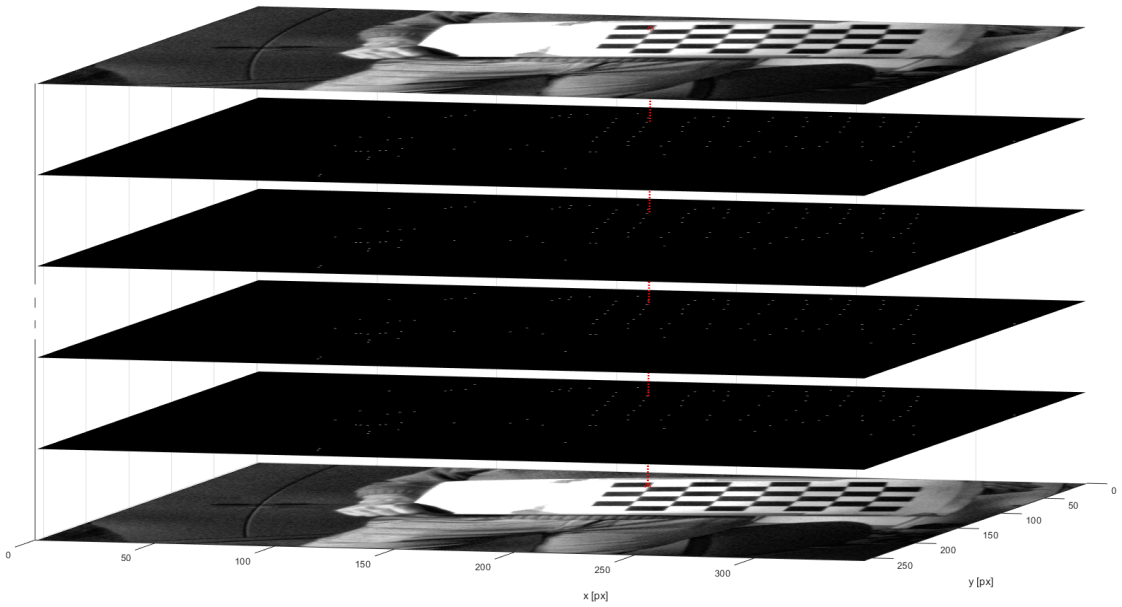


Figure 3.4: Generation of training data: the features matched between two frames are interpolated. The interpolation is then sampled at N_H equally spaced time instants in the $[t_{min}, t_{max}]$ interval.

In particular, for each set of 2000 events, t_{min} and t_{max} are defined as the timestamps of the first and last event in the set respectively. The two frames between which the interpolation is performed are then chosen to include the interval $[t_{min}, t_{max}]$, as depicted in figure 3.3.

The interval $[t_{min}, t_{max}]$ is divided into N_H equal segments, obtaining the timestamps of the heatmaps at which the interpolation is computed (figure 3.4). The locations obtained are then rounded to nearest integer since they are expressed in number of pixels.

This solution was employed considering that the dataset used is generated by a relatively smooth and slow motion, meaning the linear interpolation is a good enough approximation of the motion between two successive frames.
An example of the labels obtained is depicted in figure 3.5.

Table 3.1 contains the details of the training.

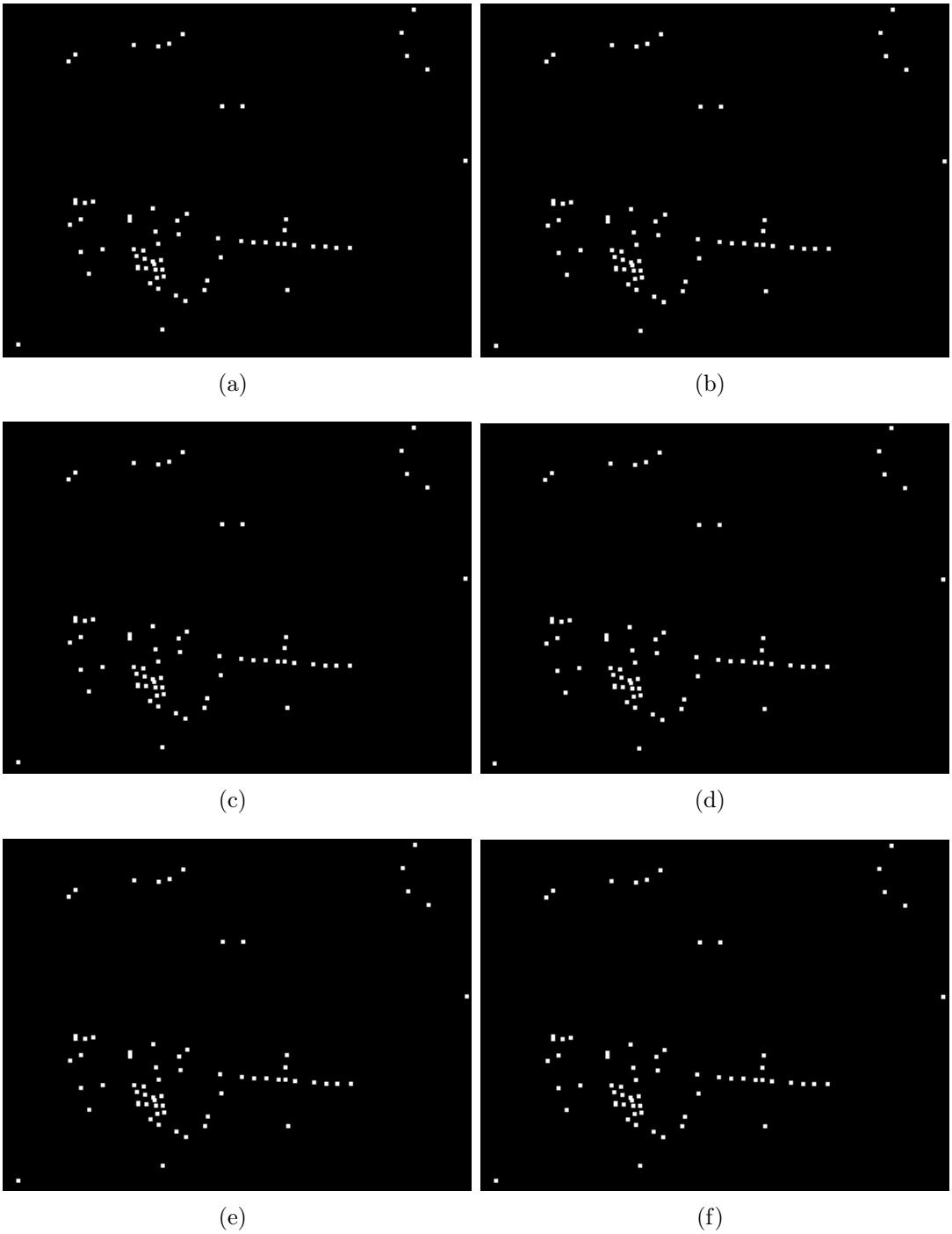


Figure 3.5: Example of a set of labels $\hat{H}$, corresponding to the event cube represented in figure 3.1. (a) to (f) represent respectively labels 1 to 6. Labels 7 to 12 not shown.

Number of epochs	Learning rate	Number of event cubes
40	0.0001	6750
Number of events	Number of keypoints	Avg number of keypoints per label
$13.5 * 10^6$	5442164	67.187

Table 3.1: Training details.

Figure 3.6: The DAVIS346 event-camera used in this work [14].

3.4 Tracking

Once the keypoints are extracted, they are matched between successive predicted frames with a nearest neighbor matching algorithm, provided that the Euclidean distance between matched points is smaller than 4 pixel. The tracks are then merged if the following conditions are verified between the last point (x_i, y_i, t_i) of one and the first point (x_j, y_j, t_j) of another:

- $d((x_i, y_i), (x_j, y_j)) < 4$ pixels, where $d(a, b)$ represents the Euclidean distance between points a and b, meaning the tracks extremities are close in space;
- $t_j - t_i > 0$, meaning the first track ends before the second begins;
- $t_j - t_i < 250$ event frames, meaning the track extremities are close in time;

A simple nearest neighbor tracking algorithm was chosen, following other work in literature, because of the extremely high temporal resolution of the predicted frames, which ensures the movement between consecutive frames is minimal.

3.5 Experimental apparatus

The data used in this work was collected with a DAVIS346 event camera (figure 3.6) by iniVation [3]. It can acquire event streams and grayscale frames simultaneously with a resolution of 346×260 pixels. The dynamic range for the DVS is 120 dB, while the APS only reaches 56.7 dB. The minimum latency is about 20 μs and the bandwidth is 20 Mevents/s. The DAVIS346 is also equipped with a 6-axis built-in IMU. Without the lens, the device's dimensions are $40 \times 60 \times 25$ mm and it weighs 100 g. Its total power consumption is lower than 180 mA @ $5V$ DC, but the DVS chip conusumes between 10 and 30 mV depending on the activity.
The DV software [15] was used to record the data from the camera and for calibration.

Results

Different acquisitions were considered, consisting of both indoor and outdoor environments. As a benchmark to validate the results obtained, eHarris was applied to the same data (in an implementation based on the description provided in [5]). Figure 4.1 presents three examples of keypoints detection. The extracted keypoints are superimposed on the closest graylevel frame just for reference, though it was not used in any way to detrmine their position. To allow for a qualitative comparison, the same 2000 events were fed into a eHarris detector and plotted on the same image.

The quality of the detected keypoints appears to be comparable to that of the keypoints detected with eHarris. It is also immediately clear how the detector presented in this work is affected by noise: while most of the keypoints are identified, in the examples provided there are many false positives generated by noise in the event streams. This problem becomes much more evident in highly textured areas. The number of events accumulated in every event cube is a key parameter: while on the one hand a longer accumulation time can help average the noise out, on the other hand it exposes to the risk of getting events cubes that are too cluttered to extract any useful information.

Figure 4.2 and figure 4.3 depict the result of the tracking process, as described in section 3.4, respectively for a checkerboard pattern and an outdoor scene. Only tracks spanning at least 20 predicted frames are included.

These examples once again prove the quality of the extracted keypoints: the tracks are coherent and follow a common motion, as expected. This is especially clear in the checkerboard case, it being a less complex image. On the other hand, the instability of the keypoints is also shown. In general the tracks are pretty short and the regularity of the checkerboard pattern allows to spot different locations in which keypoints are incorrectly merged into the same track (figure 4.4). Again, the number of accumulated events is an important parameter: figure 4.5 shows the tracking of keypoints obtained with event cubes composed of 5000 events. While the tracks are much longer, they are also much noisier.

The observations made until now are confirmed by figure 4.6 and figure 4.7. They show the number of keypoints detected, matched (between two succesive predicted frames and filtered with an MSAC algorithm to satisfy the epipolar constraint) and tracked (i.e. part of a track spanning at least 20 predicted frames in length). Table 4.1 contains the average values for these numbers. It is clear that, though a good number of keypoints is detected, the proposed algorithm struggles to track them reliably. This is probably caused by the fact that the extracted keypoints are not stable enough to be tracked with a simple nearest neighbor algorithm. The slightly better performance in the outdoor case is likely caused by the higher probability of spurious matches in a more textured environment.

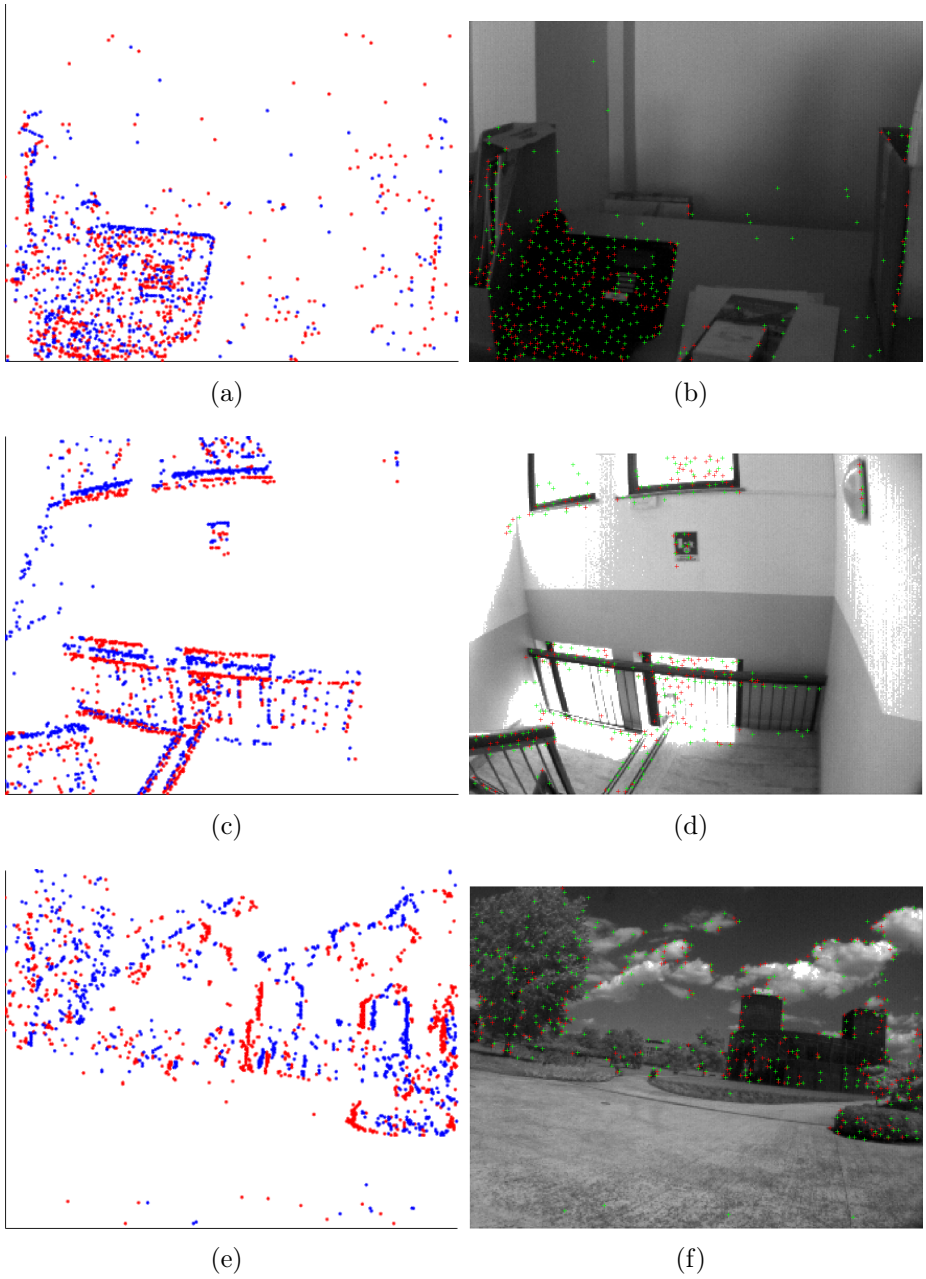


Figure 4.1: Examples of keypoint detection. (a), (c), (e) represent the event frame composed by the 2000 events used to generate the event cube (ON events in blue, OFF events in red). (b), (d), (f) represent the extracted keypoints with eHarris (red) and the method described in this work (green). For the latter, the keypints detected in the first predicted frame are depicted. Grayscale images just for reference.

	Extracted KP	Matched KP	Tracked KP	$\frac{Matched}{Extracted}$	$\frac{Tracked}{Extracted}$
Checkerboard	119.7	16.2	1.9	13.5%	1.5%
Outdoor	138.7	22.3	4.4	16.1%	3.2%

Table 4.1: Average values for the number of keypoints extracted, matched and tracked in each predicted frame.

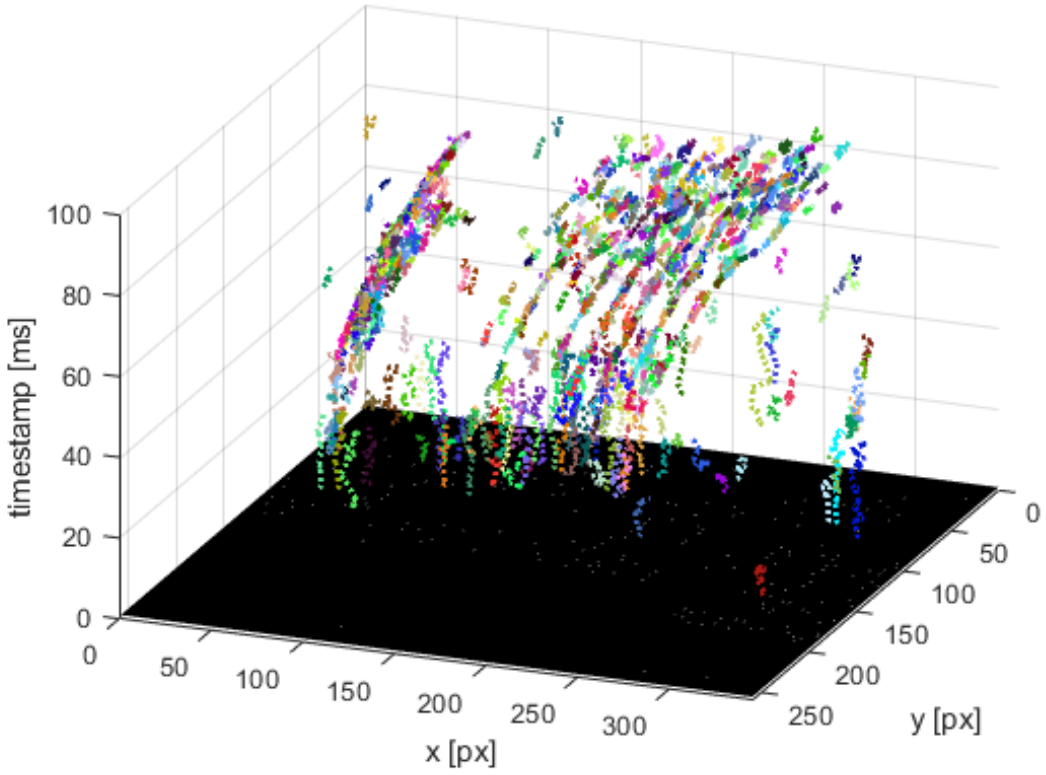


Figure 4.2: Keypoint tracking for a checkerboard pattern.

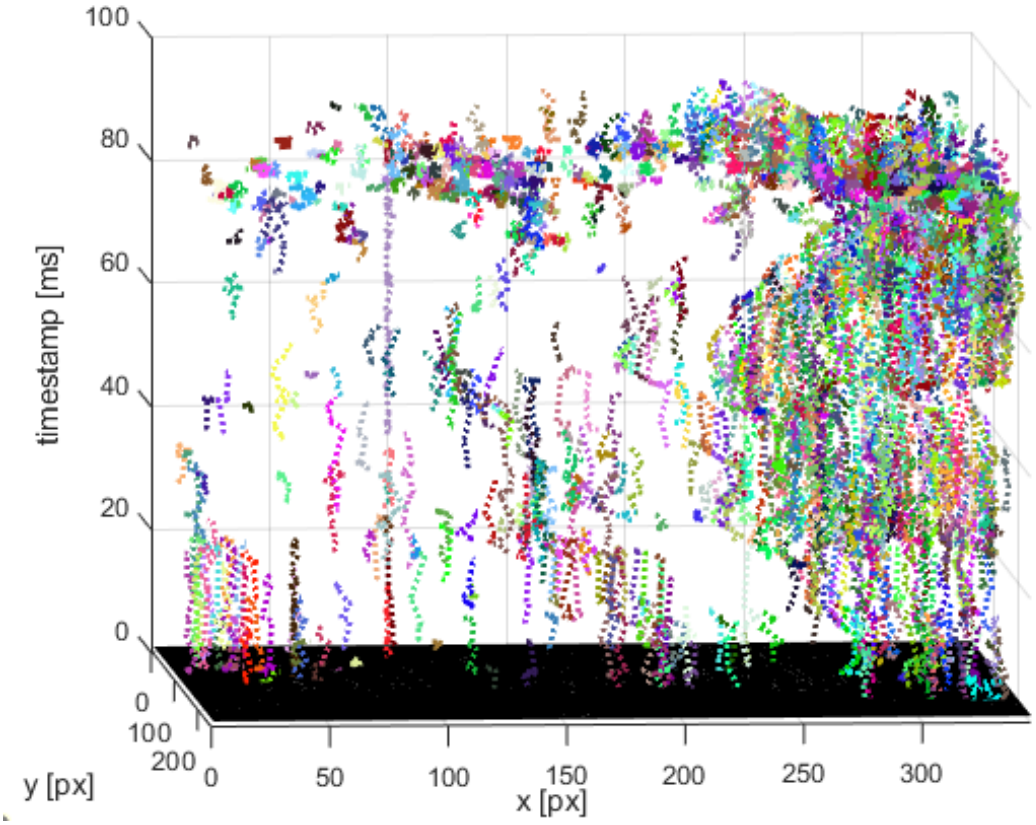


Figure 4.3: Keypoint tracking for an outdoor scene.

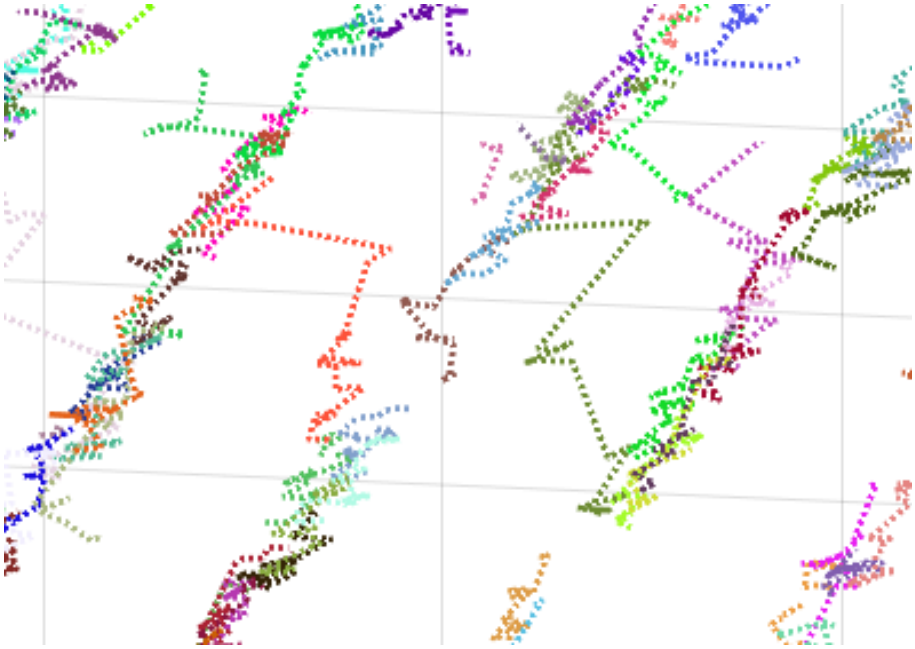


Figure 4.4: Close-up of some erroneous tracks in the checkerboard pattern. Different tracks are merged together by outlier keypoints.

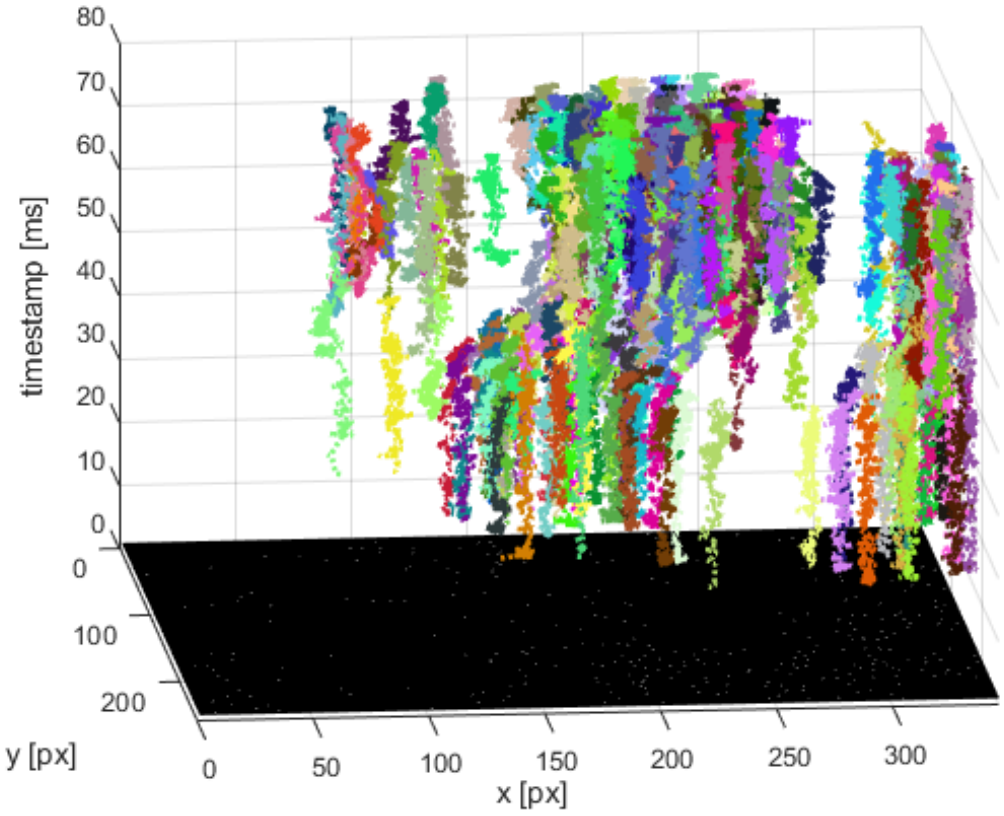


Figure 4.5: Keypoint tracking for checkerboard pattern, using $N = 5000$. The tracks are longer but also noisier.

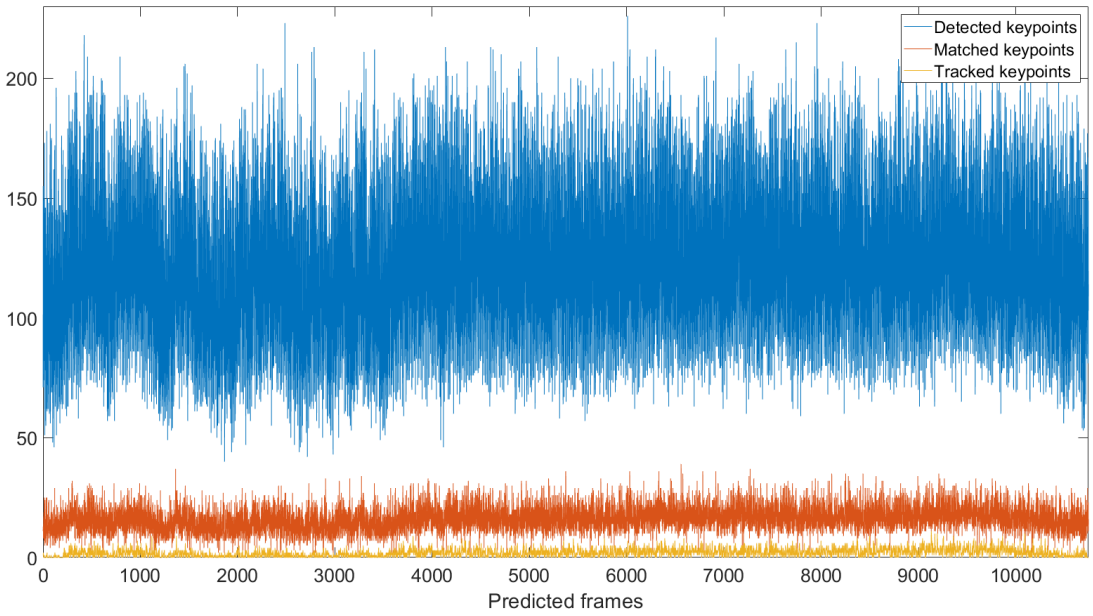


Figure 4.6: Number of keypoints detected (blue), matched (red) and tracked (yellow) in the checkerboard data.

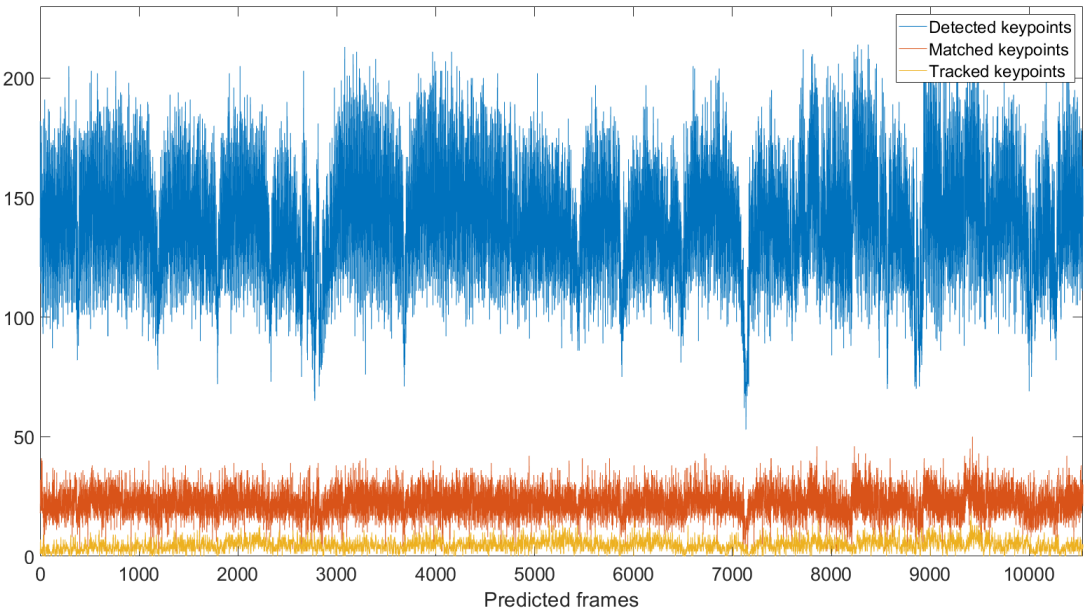


Figure 4.7: Number of keypoints detected (blue), matched (red) and tracked (yellow) in the outdoor data.

Conclusions and future development

This work presents the first steps towards a full ego-motion estimation pipeline. A novel training methodology is suggested for the keypoint detector network discussed. Though the results presented are encouraging, some work is still required to reach state of the art performance in keypoint extraction and tracking.

One first aspect that needs to be addressed is the quality of the training data: while the idea of using the grayscale frames as a reference is appealing, it is necessary to ensure that feature detection is accurate. The data used in this work presents a considerable amount of missed detections (mainly) and false positives (more rarely), which is probably the main cause for the noise that affects the extracted keypoints. While filtering with a RANSAC algorithm ensures most outliers are removed, the risk is to incorrectly label as non-keypoint pixels that are, in fact, keypoints. This, of course, has an important role in the quality of training achievable. One direction that appears interesting to explore is training the network with less, but better, data. This could mean either experimenting with different combinations of detection, matching and filtering algorithms, or even manually labelling the data, or a hybrid solution between these two. Also, since the DAVIS 346 event camera is equipped with an IMU, its data could be included as well in this process, for example through a Visual-Inertial Odometry pipeline. This would also allow to deal with different datasets, as the linear interpolation approximation wouldn't be needed anymore: the motion between two frames could be reconstructed with the IMU data. This issue should in any case be the main focus of future work.

It could be interesting to experiment with different values of N, as it appears to be a key parameter: higher values help mitigate noise, but also increase the clutter making it more difficult to select useful information among the events. In this work an example is presented where increasing N the value leads to longer but noisier tracks. This is only partially representative of the possible effects, because a network trained on 2000-event event cubes was still used. Training from the beginning with different values could yield much different results.

A different aspect that appears important to tackle is tracking: while cleaner data could be obtained as described above, a more sophisticated tracking algorithm could allow to track keypoints more reliably even with the current results. This is made especially clear by the checkerboard example: many tracks that are actually contiguous are not merged and, on the other hand, some spurious matches are allowed by outlier events. For example, it could be interesting to apply a tracking algorithm similar to the one described in [6], as it could help mitigate these issues. Also, to render the current tracks usable for ego-motion estimation, it could be useful to approximate the position of the keypoint with some kind of centroid of the

track, following [10].

Once reliable tracking is obtained, this keypoint detection algorithm could be integrated in a complete ego-motion estimation pipeline, allowing to efficiently estimate the trajectory of an autonomous vehicle. Lastly, once completed, it could be transferred to ROS (Robot Operating System) to be run in real time, given how computationally light this keypoint detection algorithm is.

Acknowledgements

The DVS/DAVIS technology was developed by the Sensors group of the Institute of Neuroinformatics (University of Zurich and ETH Zurich), which was funded by the EU FP7 SeeBetter project (grant 270324).

Bibliography

[1] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128×128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid State Circuits*, 43(2):566–576, 2008.

[2] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck. Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output. *Proceedings of the IEEE*, 102(10):1470–1484, 2014.

[3] iniVation. DAVIS 346, 08/2019. Accessed: 01 07 2023. https://inivation.com/wp-content/uploads/2019/08/DAVIS346.pdf.

[4] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 44(01):154–180, jan 2022.

[5] V. Vasco, A. Glover, and C. Bartolozzi. Fast event-based harris corner detection exploiting the advantages of event-driven cameras. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4144–4149, 2016.

[6] E. Mueggler, B. Huber, and D. Scaramuzza. Fast event-based corner detection. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1136–1141, 2017.

[7] I. Alzugaray and M. Chli. Asynchronous corner detection and tracking for event cameras in real time. *IEEE Robotics and Automation Letters*, 3(4):3177, October 2018.

[8] A. Glover, A. Dinale, L. De Souza Rosa, S. Bamford, and C. Bartolozzi. luvharris: A practical corner detector for event-cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[9] I. Alzugaray and M. Chli. Haste: multi-hypothesis asynchronous speeded-up tracking of events. In *British Machine Vision Conference*, 2020.

[10] S. Hu, Y. Kim, H. Lim, A. J. Lee, and H. Myung. ecdt: Event clustering for simultaneous feature detection and tracking. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3808–3815, 2022.

[11] J. Manderscheid, A. Sironi, N. Bourdis, D. Migliore, and V. Lepetit. Speed invariant time surface for learning to detect corner points with event-based cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 4609–4618. IEEE, 2021.

[12] A. Renner, L. Supic, A. Danielescu, G. Indiveri, B. A. Olshausen, Y. Sandamirskaya, F. T. Sommer, and E. P. Frady. Neuromorphic visual scene understanding with resonator networks, 2022.

[13] P. Chiberre, E. Perot, A. Sironi, and V. Lepetit. Detecting stable keypoints from events through image gradient prediction. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1387–1394, 2021.

[14] . . Accessed: 04 07 2023. https://blog.csdn.net/tfb760/article/details/119637494.

[15] iniVation. dv-core, 2023. Accessed: 04 07 2023. https://gitlab.com/inivation/dv.

[16] P. Lichtsteiner and T. Delbruck. A 64x64 aer logarithmic temporal derivative silicon retina. In *Research in Microelectronics and Electronics, 2005 PhD*, volume 2, pages 202–205, 2005.

[17] 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 Journal of Solid-State Circuits*, 46(1):259–275, 2011.

[18] R. Berner, C. Brandli, M. Yang, S.-C. Liu, and T. Delbruck. A 240 x 180 10 mw 12 μs latency sparse-output vision sensor for mobile applications. *IEEE Journal of Solid-State Circuits*, 48(5):1340–1351, 2013.

[19] ESA - Advanced Concepts Team. Retinomorphic Vision Model for On-chip Feature Extraction. Accessed: 04 07 2023. https://www.esa.int/gsp/ACT/projects/retinomorphic$_vision$/.

[20] O. Sikorski, D. Izzo, and G. Meoni. Event-based spacecraft landing using time-to-contact. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1941–1950, 2021.

[21] M. Wudenka, M. G. Müller, N. Demmel, A. Wedler, R. Triebel, D. Cremers, and W. Stürzl. Towards robust monocular visual odometry for flying robots on planetary missions. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8737–8744, 2021.

[22] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision (ECCV)*, pages 430–443, 2006.

[23] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi. Event-based visual flow. *IEEE Transactions on Neural Networks and Learning Systems*, 25(2):407, February 2014.

[24] I. Alzugaray and M. Chli. Asynchronous multi-hypothesis tracking of features with event cameras. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[25] I. Alzugaray and M. Chli. Asynchronous multi-hypothesis tracking of features with event cameras. In *2019 International Conference on 3D Vision (3DV)*, pages 269–278, 2019.

[26] A. Renner, L. Supic, A. Danielescu, G. Indiveri, E. P. Frady, F. T. Sommer, and Y. Sandamirskaya. Neuromorphic visual odometry with resonator networks, 2022.

[27] E. P. Frady, S. J. Kent, B. A. Olshausen, and F. T. Sommer. Resonator networks, 1: An efficient solution for factoring high-dimensional, distributed representations of data structures. *Neural computation*, 32(12):2311–2331, Dec 2020.

[28] S. J. Kent, E. P. Frady, F. T. Sommer, and B. A. Olshausen. Resonator networks, 2: Factorization performance and capacity compared to optimization-based methods. *Neural computation*, 32(12):2332–2388, 2020.

[29] E. Paxon Frady, Denis Kleyko, Christopher J. Kymn, Bruno A. Olshausen, and Friedrich T. Sommer. Computing on functions using randomized vector representations (in brief). In *Proceedings of the 2022 Annual Neuro-Inspired Computational Elements Conference*, page 115–122, New York, NY, USA, 2022. Association for Computing Machinery.

[30] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza. Events-to-video: Bringing modern computer vision to event cameras. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3852–3861, 2019.

[31] P. Chiberre, E. Perot, A. Sironi, and V. Lepetit. Long-lived accurate keypoints in event streams. *ArXiv*, abs/2209.10385, 2022.

[32] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 989–997, 2018.

[33] C. Scheerlinck, H. Rebecq, D. Gehrig, N. Barnes, R. Mahony, and D. Scaramuzza. Fast image reconstruction with an event camera. In *IEEE Winter Conference on Applications of Computer Vision*, pages 156–163, 2020.

[34] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu. Squeeze-and-excitation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(8):2011–2023, 2020.

[35] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 802–810, Cambridge, MA, USA, 2015. MIT Press.