University of Zurich[UZH]

Department of Informatics

Event-based Vision for High-Speed Robotics

Dissertation
submitted to the
Faculty of Business, Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor / Doktorin der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by

Elias Mueggler
from Fischingen TG

approved in July 2017 at the request of
Prof. Dr. Davide Scaramuzza, advisor
Prof. Dr. Tobi Delbruck, examiner
Prof. Dr. Kostas Daniilidis, examiner

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, 19.07.2017

Chairwoman of the Doctoral Board: Prof. Dr. Elaine M. Huang

To my family and Michelle.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Davide Scaramuzza for selecting me as a PhD student. Davide's vision of robotics inspired me, he provided me with plenty of exciting opportunities, and he also gave me the freedom to pursue my own ideas. It was a rewarding experience to see the lab grow in the last years from just a few to more than a dozen staff members.
Collaboration enjoys a very high priority in the Robotics and Perception Group and indeed, this thesis would not have been possible without the help, fruitful discussions, and fun distractions from my colleagues. I therefore wish to express my gratitude to all the current and past members, visitors, and students. I would particularly like to thank Matthias Faessler, Flavio Fontana, Christian Forster, Davide Falanga, Henri Rebecq, Titus Cieslewski, Zichao Zhang, Junjie Zhang, Michael Gassner, Alessandro Simovic, Raphael Meyer, Guillermo Gallego, Jeff Delmerico, Matia Pizzoli, Andras Majdik, Manuel Werlberger, Antonio Toma, Gabriele Costante, Volker Grabe and Tamar Tolcachier. I also had the pleasure to work with great students, namely Karl Schwabe, Nathan Baumli, Basil Huber, Julia Nitsch, Micha Brunner, Beat Kueng, Mathis Kappeler, Jon Lund, Timo Horstschäfer, Benjamin Keiser and David Tedaldi.
I am very grateful to Chiara Bartolozzi for inviting me to a very inspiring stay at the Istituto Italiano di Tecnologia in Genoa, Italy, and the NCCR International PhD/Postdoc Exchange Programme for the scholarship. I would also like to thank Valentina Vasco and Arren Glover for their great support and interesting discussions.
I would like to thank the agencies funding my research, namely the National Centre of Competence in Research (NCCR) Robotics, the Swiss National Science Foundation, the DARPA FLA Program, and Google. I would also like to thank Qualcomm for their support through the Qualcomm Innovation Fellowship and Gerhard Reitmayr who was a great mentor throughout this last year. I would like to thank Sim Bamford and Luca Longinotti from iniLabs for their fast and effective support with event-camera software and hardware.
I would like to thank Prof. Kostas Daniilidis and Prof. Tobi Delbruck for accepting to review my thesis and for their valuable feedback.
Last but not least, I am very grateful to my family, Michelle, and my friends who supported me at all times.

Zurich, July 2017 E. M.

Abstract

Cameras are appealing sensors for mobile robots because they are small, passive, inexpensive and provide rich information about the environment. While cameras have been used successfully on a plenitude of robots, such as autonomous cars or drones, serious challenges remain: power consumption, latency, dynamic range, and frame rate, among others. The sequences of images acquired by a camera are highly redundant (both in space and time), and both acquiring and processing such an amount of data consumes significant power. This limits the operation time of mobile robots and, moreover, defines a fundamental power–latency tradeoff. Specialized cameras designed for high-speed or high-dynamic–range scenarios are expensive, heavy, and require additional power, which prevents their use in agile mobile robots.

In this thesis, we investigate *event cameras* as a biologically-inspired alternative to overcome the limitations of standard cameras. These neuromorphic vision sensors work in a completely different way: instead of providing a sequence of images (i.e., frames) at a constant rate, event cameras transmit only information from those pixels that undergo a significant brightness *change*. These pixel-level brightness changes, called *events*, are timestamped with *micro*-second resolution and transmitted asynchronously at the time they occur. Hence, event cameras are power efficient because they convey only non-redundant information, and are able to capture very high-speed motions, thus they directly address the power–latency tradeoff. Additionally, event cameras achieve a dynamic range of more than 140 dB, compared to about 60 dB of standard cameras, because each pixel is autonomous and operates at its own set-point. However, since the output of an event camera is fundamentally different from that of standard cameras for which computer-vision algorithms have been developed during the past fifty years, new algorithms that can deal with the asynchronous nature of the sensor and exploit its high temporal resolution are required to unlock its potential.

This thesis presents algorithms for using event cameras in the context of robotics. Since event cameras are novel sensors that are being intensively prototyped and have been commercially available only recently (ca. 2008), the literature on event-based algorithms is scarce. This poses some operational challenges as well as uncountable opportunities to explore in research. This thesis focuses on exploring the possibilities that event cameras bring to some fundamental problems in robotics and computer vision, such as localization and actuation. Amongst others, this thesis provides contributions to solving the localization problem, i.e., for a robot equipped with an event camera to be able to infer its location with respect to a given map of the environment. Classical approaches for robot localization build upon lower-level vision algorithms, and so, this thesis also presents contributions in the topics of detection, extraction, and tracking of

salient visual features with an event camera, whose applicability expands far beyond the localization problem. This thesis also presents contributions in the use of event cameras for actuation and closed-loop control, i.e., in endowing the robot with the capabilities to interact with the environment to fulfill a given task. Additionally, this thesis also presents the infrastructure developed to work with event cameras in a de-facto standard robotics platform. The following is a list of contributions:

- Software infrastructure, consisting of publicly available drivers, calibration tools, sensor delay characterization, and the first event camera dataset and simulator tailored for 6-DOF (degrees of freedom) camera pose estimation and SLAM (Simultaneous localization and mapping).
- We introduce the concept of event "lifetime" and provide an algorithm to compute it. The lifetime endows the events with a finite temporal extent for a proper continuous representation of events in time.
- The first method to extract FAST-like visual features (i.e., interest points or corners) from the output of an event camera. The detector operates an order of magnitude faster than previous corner detectors.
- The first method to extract and track features from the output of a DAVIS camera (an event camera that also outputs standard frames from the same pixel array). Using these feature tracks, we developed the first sparse, feature-based visual-odometry pipeline.
- The first two methods to track the 6-DOF pose of an event camera in a known map. While the first method minimizes the reprojection error of the events and only works on black-and-white scenes consisting of line segments, the second method uses a probabilistic filtering framework that allows tracking at high speeds on natural scenes.
- The first application of a continuous-time framework to estimate the trajectory of an event camera, possibly incorporating inertial measurements, showing superior performance over pose–tracking-only methods.
- An application of event cameras to collision avoidance of a quadrotor, showing how event cameras can be used to control a robot with very low latency.
- An application of the use of an event camera for human-vs-machine slot-car racing, showing that event-driven algorithms are power efficient and can outperform human control.

Zusammenfassung

Kameras sind sehr nützliche Sensoren für mobile Roboter, weil sie klein, passiv und kostengünstig sind sowie reichhaltige Informationen der Umgebung liefern. Obwohl Kameras erfolgreich in einer Vielzahl von Robotern, wie zum Beispiel autonomen Fahrzeugen oder Drohnen, verwendet werden, stellen Energiebedarf, Latenz, Dynamikbereich und Bildfrequenz beträchtliche Herausforderungen dar. Die Bildsequenz von Kameras enthält viel Redundanz (sowohl zeitlich wie räumlich) und sowohl das Aufnehmen wie das Verarbeiten dieser Datenmenge benötigt viel Leistung. Dies limitiert die Betriebszeit mobiler Roboter und definiert einen fundamentalen Kompromiss zwischen Energiebedarf und Latenz. Spezialkameras für Hochgeschwindigkeits- und Hochkontrastanwendungen sind teuer, schwer, und brauchen zusätzliche Energie, was deren Anwendung in agilen mobilen Robotern verunmöglicht.

In dieser Dissertation untersuchen wir *Event-Kameras* als bioinspirierte Alternative um die Limitationen von Standardkameras zu überwinden. Diese neuromorphischen visuellen Sensoren funktionieren auf komplett andere Weise. Anstatt einer Bildsequenz mit einer konstanten Frequenz zu liefern, senden Event-Kameras nur Informationen von den Pixeln, bei denen sich die Helligkeit signifikant verändert hat. Solche pixelweise Veränderungen nennen wir *Events*, welche mit einem Zeitstempel mit der Genauigkeit von *Mikro*-Sekunden versehen und unmittelbar danach asynchron übermittelt werden. Da nur nicht-redundante Informationen übertragen werden sind Event-Kameras energieeffizient und in der Lage, sehr schnelle Bewegungen zu erfassen. Damit nehmen sie den Kompromiss zwischen Energiebedarf und Latenz direkt in Angriff. Zudem verfügen Event-Kameras über einen Dynamikbereich von über 140 dB (Standardkameras verfügen typischerweise um die 60 dB), weil jedes Pixel selbständig ist. Da das Datensignal einer Event-Kamera fundamental anders ist als dasjenige einer Standardkamera (für welche über die letzten fünfzig Jahren Algorithmen für maschinelles Sehen entwickelt wurden) werden neue Algorithmen benötigt, die mit der asynchronen Funktionsweise klarkommen und die hohe zeitliche Auflösung ausnutzen können.

Diese Dissertation präsentiert Algorithmen für Event-Kameras im Bereich Robotik. Da Event-Kameras neuartige Sensoren sind und kommerziell erst seit 2008 erhältlich sind, ist die Literatur über solche Algorithmen spärlich. Dies erschwert die Handhabung dieser Sensoren, eröffnet aber unzählige Möglichkeiten, die es zu erforschen gilt. Diese Dissertation untersucht die Möglichkeiten von Event-Kameras für fundamentale Probleme der Robotik und des maschinellen Sehens wie zum Beispiel Lokalisierung und Steuerung. Unter anderem bietet diese Dissertation Beiträge zur Lösung des Lokalisierungsproblems, d.h. für einen Roboter, der mit einer Event-Kamera ausgestattet ist, in der Lage zu sein, seinen Standort bezüglich einer gegebenen Karte der Umgebung zu

bestimmen. Klassische Ansätze zur Roboterlokalisierung bauen auf untergeordneten Algorithmen auf, sodass diese Dissertation auch Beiträge zu den Themen Detektion, Extraktion und Verfolgung von markanten visuellen Merkmalen (Features) mit einer Event-Kamera präsentiert, deren Anwendbarkeit weit über das Lokalisierungsproblem hinausgeht. Diese Arbeit präsentiert auch Beiträge zur Verwendung von Event-Kameras für die Steuerung und Regelung, d.h. der Möglichkeit eines Roboters mit seiner Umgebung zu interagieren um ein bestimmtes Ziel zu erreichen. Darüber hinaus präsentiert diese Dissertation auch die Infrastruktur, die entwickelt wurde, um Event-Kameras in einer weitverbreiteten Robotikplattform zu verwenden. Es folgt eine Liste der Beiträge:

- Software-Infrastruktur, bestehend aus öffentlich zugänglichen Treibern, Kalibrierungswerkzeugen, Charakterisierung der Sensorlatenz und dem ersten Datensatz und Simulator von Event-Kameras für Kamerapositionsschätzung und SLAM (Simultane Lokalisierung und Kartierung) mit sechs Freiheitsgraden (FHG).
- Wir stellen das Konzept der Event-"Gültigkeitsdauer" vor und liefern einen Algorithmus um diese zu berechnen. Die Gültigkeitsdauer verleiht einem Event eine endliche zeitliche Ausdehnung und erlaubt eine kontinuierliche Darstellung von Events in der Zeit.
- Die erste Methode um FAST-ähnliche visuelle Features (d.h. charakteristische Punkte oder Ecken) aus dem Datensignal einer Event-Kamera zu extrahieren. Der Detektor läuft eine Grössenordnung schneller als bisherige Detektoren.
- Die erste Methode um Features aus dem Datensignal einer DAVIS-Kamera (eine Event-Kamera, die nebst Events auch normale Bilder von den gleichen Pixeln ausgibt) zu extrahieren und zu verfolgen. Mit diesen Features entwickelten wir das erste Feature-basierte visuelle Odometrie-System.
- Die ersten beiden Methoden, um die Bewegung einer Event-Kamera mit 6 FHG in einer bekannten Karte der Umgebung zu schätzen. Die erste Methode minimiert den Reprojektionsfehler der Events und funktioniert nur auf Schwarz-Weiss-Szenen, die aus Liniensegmenten bestehen. Die zweite Methode hingegen verwendet einen probabilistischen Filter, der die Verfolgung bei hohen Geschwindigkeiten in natürlicher Umgebung ermöglicht.
- Die erste Anwendung einer zeitkontinuierlichen Darstellung der Trajektorie einer Event-Kamera, die ebenfalls Inertialmessungen beinhaltet kann, die Filter-basierten Methoden übertrifft was Genauigkeit betrifft.
- Eine Anwendung von Event-Kameras zur Kollisionsvermeidung eines Quadrokopters, die zeigt, wie Event-Kameras verwendet werden können um einen Roboter mit sehr geringer Latenz zu steuern.
- Eine Anwendung von einer Event-Kamera für Mensch-gegen-Maschine-Rennen auf einer Modellautorennbahn zeigt, dass Event-basierte Algorithmen effizient sind und die menschliche Leistung übertreffen können.

List of Contributions

Journal Publications

- Guillermo Gallego, Jon E. A. Lund, **Elias Mueggler**, Henri Rebecq, Tobi Delbruck, and Davide Scaramuzza. "Event-based, 6-DOF Camera Tracking for High-Speed Applications". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017). Under review.
  Links: Appendix H, PDF, Video

- Henri Rebecq, Guillermo Gallego, **Elias Mueggler**, and Davide Scaramuzza. "EMVS: Event-based Multi-View Stereo". In: *International Journal of Computer Vision* (2017). Under review.

- **Elias Mueggler**, Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. "Continuous-Time Visual-Inertial Trajectory Estimation with Event Cameras". In: *IEEE Transactions on Robotics* (2017). Under review.
  Links: Appendix I, PDF

- **Elias Mueggler**, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. "The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM". In: *International Journal of Robotics Research* 36 (2 2017), pp. 142–149. DOI: 10.1177/0278364917691115
  Links: Appendix C, PDF, Video, Dataset

- Jeffrey Delmerico, **Elias Mueggler**, Julia Nitsch, and Davide Scaramuzza. "Active Autonomous Aerial Exploration for Ground Robot Path Planning". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 664–671. DOI: 10.1109/LRA.2017.2651163
  Links: PDF, Video

- Matthias Faessler, Flavio Fontana, Christian Forster, **Elias Mueggler**, Matia Pizzoli, and Davide Scaramuzza. "Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor MAV". In: *Journal of Field Robotics* 33.4 (2016), pp. 431–450. DOI: 10.1002/rob.21581
  Links: PDF, Video 1, Video 2, Video 3, Video 4, Software

Peer-Reviewed Conference Papers

- **Elias Mueggler**, Chiara Bartolozzi, and Davide Scaramuzza. "Fast Event-based Corner Detection". In: *British Machine Vision Conference (BMVC)*. 2017
  Links: Appendix E, Video

- Valentina Vasco, Arren Glover, Lorenzo Natale, Chiara Bartolozzi, **Elias Mueggler**, and Davide Scaramuzza. "Independent motion detection with event-driven cameras". In:

International Conference on Advanced Robotics (ICAR). 2017.
Links: PDF,

- Davide Falanga, **Elias Mueggler**, Matthias Faessler, and Davide Scaramuzza. "Aggressive Quadrotor Flight through Narrow Gaps with Onboard Sensing and Computing". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017
  Links: PDF, Video

- Roman Käslin, Péter Fankhauser, Elena Stumm, Zachary Taylor, **Elias Mueggler**, Jeffrey Delmerico, Davide Scaramuzza, Roland Siegwart, and Marco Hutter. "Collaborative localization of aerial and ground robots through elevation maps". In: *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. 2016, pp. 284–290. DOI: 10.1109/SSRR.2016.7784317
  Links: PDF

- Beat Kueng, **Elias Mueggler**, Guillermo Gallego, and Davide Scaramuzza. "Low-latency Visual Odometry using Event-based Feature Tracks". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 16–23. doi: DOI: 10.1109/IROS.2016.7758089
  Links: Appendix G, PDF, Video

- David Tedaldi, Guillermo Gallego, **Elias Mueggler**, and Davide Scaramuzza. "Feature Detection and Tracking with the Dynamic and Active-pixel Vision Sensor (DAVIS)". In: *International Conference on Event-Based Control, Communication and Signal Processing (EBCCSP)*. 2016, pp. 1–7. DOI: 10.1109/EBCCSP.2016.7605086
  Links: Appendix F, PDF, Video

- Jeffrey Delmerico, Alessandro Giusti, **Elias Mueggler**, Luca Maria Gambardella, and Davide Scaramuzza. ""On-the-spot Training" for Terrain Classification in Autonomous Air-Ground Collaborative Teams". In: *International Symposium on Experimental Robotics (ISER)*. 2016. pp. 574–585. DOI: 10.1007/978-3-319-50115-4_50
  Links: PDF, Video

- **Elias Mueggler**, Nathan Baumli, Flavio Fontana, and Davide Scaramuzza. "Towards Evasive Maneuvers with Quadrotors using Dynamic Vision Sensors". In: *European Conference on Mobile Robotics (ECMR)*. 2015, pp. 1–8. DOI: 10.1109/ECMR.2015.7324048
  Links: Appendix B, PDF

- **Elias Mueggler**, Guillermo Gallego, and Davide Scaramuzza. "Continuous-Time Trajectory Estimation for Event-based Vision Sensors". In: *Robotics: Science and Systems (RSS)*. 2015. DOI: 10.15607/RSS.2015.XI.036
  Links: Appendix I, PDF

- **Elias Mueggler**, Christian Forster, Nathan Baumli, Guillermo Gallego, and Davide Scaramuzza. "Lifetime Estimation of Events from Dynamic Vision Sensors". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 4874–4881. DOI: 10.1109/ICRA.2015.7139876
  Links: Appendix D, PDF

- Tobi Delbruck, Michael Pfeiffer, Raphaël Juston, Garrick Orchard, **Elias Müggler**, Alejandro Linares-Barranco, and Mark W. Tilden. "Human vs. Computer Slot Car Racing using an Event and Frame-Based DAVIS Vision Sensor". In: *International Symposium on Circuits*

and Systems (ISCAS). 2015, pp. 2409–2412. DOI: 10.1109/ISCAS.2015.7169170
Links: Appendix J, PDF, Video 1, Video 2, Software

- **Elias Mueggler**, Matthias Faessler, Flavio Fontana, and Davide Scaramuzza. "Aerial-guided Navigation of a Ground Robot among Movable Obstacles". In: *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. 2014, pp. 1–8. DOI: 10.1109/SSRR.2014.7017662
Links: PDF, Video 1, Video 2

- **Elias Mueggler**, Basil Huber, and Davide Scaramuzza. "Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2014, pp. 2761–2768. DOI: 10.1109/IROS.2014.6942940
Links: Appendix A, PDF, Video

- Matthias Faessler, **Elias Mueggler**, Karl Schwabe, and Davide Scaramuzza. "A Monocular Pose Estimation System based on Infrared LEDs". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 907–913. doi: DOI: 10.1109/ICRA.2014.6906962
Links: PDF, Video

Awards

- **Misha Mahowald Prize for Neuromorphic Engineering ($3.000), 2017**
- NCCR Robotics PhD/Postdoc Exchange Grant, 2016
- IROS Best Application Paper Award Finalist, 2016
- **Qualcomm Innovation Fellowship Europe ($40.000), 2016**
- SSRR Best Paper Award Finalist, 2014
- **Winner of KUKA Innovation Award (€20.000), 2014**
- Convergent Science Network of Biomimetics and Neurotechnology CapoCaccia Fellowship, 2013

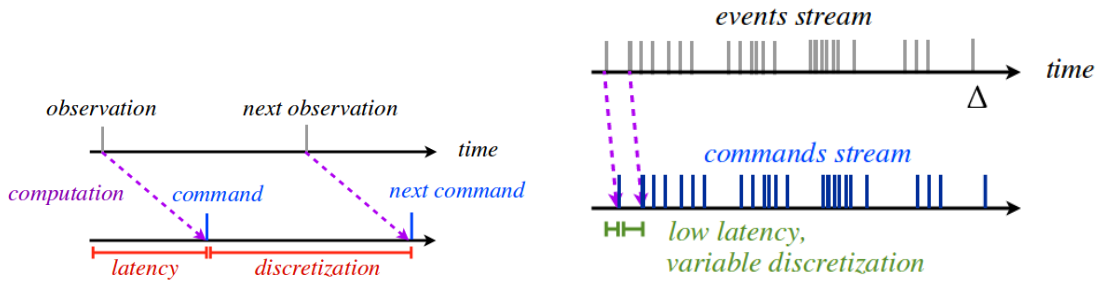
Contents

1 Introduction

> Conventional scanning techniques require that each node be sampled once every frame. Since the retina generates output only at areas in the image where there is spatial or temporal change in the image, most of the nodes will have almost no output, but are sampled anyway. The address-event protocol, in contrast, is data driven. Only pixels that have something to report are transmitting their output over the data bus. Therefore, areas of uniform illumination do not contribute to the communication load. A further major advantage of the address-event communications framework is that it minimizes temporal aliasing by transmitting events as they occur. It need not introduce the degree of sampling inherent in a sequential scanning technique.
>
> *Misha Mahowald, 1992 [90]*

This thesis presents algorithms to process data from event cameras in the context of high-speed robotics. In contrast to standard cameras that capture images at a fixed rate, event cameras have independent, asynchronous pixels that report local brightness *changes* (called "events") at the time they occur, thus mimicking the above-mentioned operating principle of the retina [90]. In addition to the high temporal resolution and low latency, both in the order of *micro*-seconds, event cameras also provide a large dynamic range (140 dB compared to 60 dB of standard cameras). However, since the output of event cameras is fundamentally different (an asynchronous stream of events rather than frames), new algorithms are required to deal with these data.

This thesis is split into five parts: first, infrastructure for event cameras in robotics is provided, such as drivers, a calibration toolbox, sensor characterization, a simulator, and a large collection of publicly available datasets. Second, event-based feature detection algorithms are described that augment the event stream with additional information (e.g., their lifetime or whether they are a corner event). Third, event-based tracking algorithms for low-level features—such as polygonal shapes, balls, or arbitrary corners—are presented. Fourth, methods for event-based ego-motion estimation are presented. Finally, applications of event cameras in closed-loop robotic control are presented.



(a) Traditional architecture **(b)** Event-based architecture

Figure 1.1: Comparison of the output of a standard camera and an event camera when used for robot control: the low latency of event cameras allow much faster reactions to changes in the environment. Figure from [27].

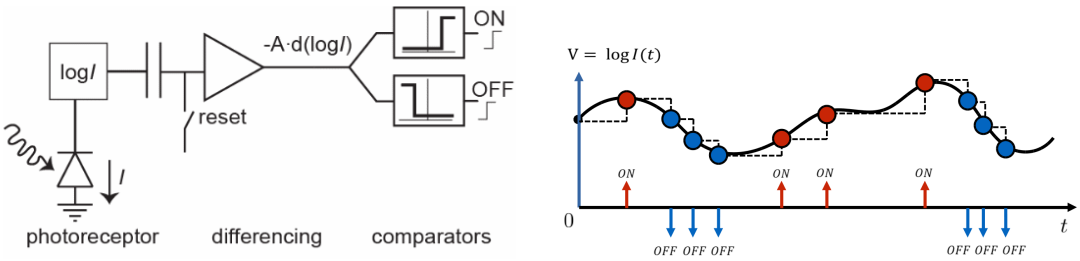
This thesis is structured in the form of a collection of papers. An introductory section that highlights the concepts and ideas behind the thesis is followed by self-contained publications in the appendix.

In the next section, the working principle, advantages, and challenges of event cameras are discussed. Section 1.2 motivates and states the research objectives of this dissertation. The papers in the appendix are summarized in Chapter 2. Finally, Chapter 3 provides future research directions.

1.1 Event Cameras for Mobile Robots

Mobile robots need to perceive their environment to navigate safely in it. To make the robot independent of its environment, it should carry all sensors. For agile and fast robots, large and heavy sensors such as laser scanners are not suitable. Using vision as primary sensing modality provides several advantages. Cameras are small, lightweight, and passive, thus they require little power. Yet, they provide rich information about the environment. Interpreting these data is, however, a challenging task that consumes a significant amount of computational resources and power. For comparison, more than 60 % of the human brain is involved in vision-related tasks. In the past decades, tremendous progress in computer vision has been achieved, even exceeding human performance on certain tasks. These algorithms are used for face detection and identification, image labeling, and many other tasks. In the domain of robotics, we are primarily interested in mapping and localization.

While standard cameras offer many advantages, severe challenges remain. First, at high speeds, images suffer from motion blur that will cause subsequent algorithms to fail. Second, cameras provide a rather low intra-scene dynamic range, thus in scenes with large illumination variations certain regions in the image will be under-



(a) Basic Analog Circuit **(b)** Response to Intensity Profile

Figure 1.2: Event Camera Pixels. Figures adapted from [77].

or over-exposed, with faded information. Third, the frame rate of cameras is limited and thus provides an upper bound on the achievable latency of a robotic system (see Fig. 1.1). Fourth, the higher the frame rate (a parameter that is independent of the variability of the visual information in the scene), the more data needs to be processed, and so, the more power is consumed in, typically, useless operations since images are highly redundant. To overcome these limitations, we investigate the use of event cameras in the domain of robotics. Next, we introduce event cameras as well as their advantages and challenges.

1.1.1 Working Principle

Event cameras have independent, asynchronous pixels that report local brightness changes at the time they occur. Since they are inspired by the retina of vertebra, event cameras are also called "silicon retinas". The circuit of a single pixel is shown in Fig. 1.2a. Its output is a stream of events, whenever the (logarithmic) brightness changes by a user-defined threshold (cf. Fig. 1.2b). To illustrate their working principle for the entire sensor, we compare the output of an event camera to the one of a standard camera in Fig. 1.3 and in a online video[1]. Both cameras are observing a rotating disk with a black circle in its perifery. A standard camera outputs a video, i.e., a series of frames at a constant rate. Instead, an event camera only reports the brightness changes.

1.1.2 Advantages

In this section, we summarize the main advantages of event cameras compared to standard frame-based cameras.

[1] https://youtu.be/LauQ6LWTkxM

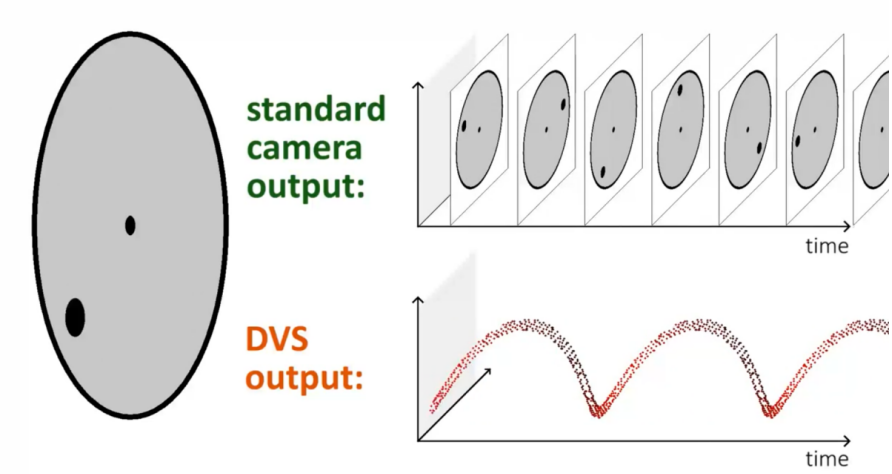


Figure 1.3: Comparison of the outputs of a standard camera and an event camera when observing a rotating disk with a black circle: the standard camera outputs a series of frame, while the event camera continuously and asynchronously reports the *changes* in the scene. Each dot in space-time represents such a change (called "event"). If the disk stops rotating, no events are transmitted, whereas a standard camera continues to transmit the same redundant frame. High-speed rotation of the disk will cause motion blur on the images. However, due to the low latency of the event camera, the same space-time helix is observed for faster rotation—only squeezed in the time axis.

Low Latency and High Temporal Resolution. Event cameras provide very low latency to input changes, which are in the order of a few microseconds: the DVS [77] and DAVIS [19] have latencies of 15 µs and 3 µs, respectively. As comparison, the latency of frame-based sampling is a uniform distribution between 0 and $1/f$, where f is the temporal discretization (frame rate). For a standard camera at $f = 30$ Hz, the temporal discretization can be as high as 33 ms—four orders of magnitude higher! Therefore, event cameras allow much faster control loops (see Fig. 1.1).

High Dynamic Range. Since each pixel of an event camera is independent and chooses its own set-point, event cameras reach very high intra-scene dynamic ranges: the DVS [77], DAVIS [19], and ATIS [116] achieve dynamic ranges of 120 dB, 130 dB, and 143 dB, respectively. Machine-vision cameras typically achieve 60 dB. Therefore, event cameras can still be used in scenes with very bright and very dark regions (see Fig. 1.4 for an extreme example).

Low Bandwith. Because event cameras only report pixel-level brightness changes, no bandwidth is required if pixels do not change value, i.e., if there is no relative motion between the scene and the camera or there are no illumination changes. Furthermore, the events originate from salient regions (intensity edges) and, therefore, many preprocessing operations needed for frames become unnecessary. This allows fast reaction at

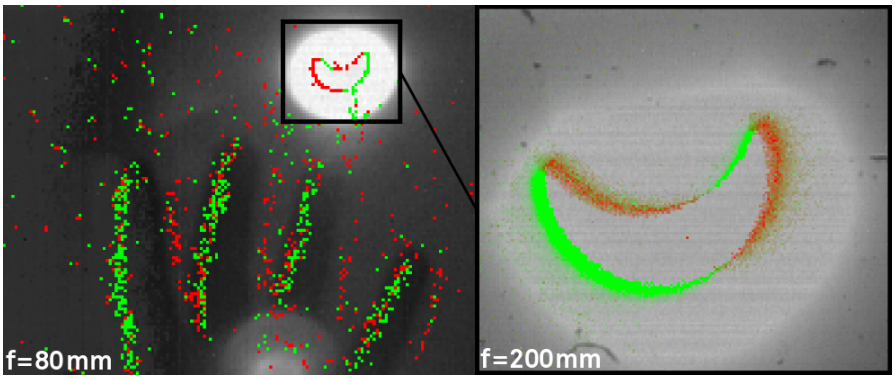


Figure 1.4: Solar eclipse seen by a standard and an event camera. While the image exhibits both overexposed and underexposed areas, the events can detect the contours of the eclipse flawlessly. Image courtesy of Mark Osswald, Simeon Bamford and Tobi Delbruck.

low processing cost (see Fig. 1.1b).

Low Power. An event camera requires less power compared to a standard camera. The main reason is that analog-to-digital converters, which are required for pixel readout, consume relatively much power. Such converters are not required in event cameras. However, both event cameras and standard cameras require only power in the range of a few mW. Much more power can be saved when considering the entire system (sensor and processing). As Prof. Marc Pollefeys noted regarding an augmented-reality system based on standard cameras:[2] "Most of the energy is spent moving bits around . . . so it would seem natural that . . . the first layers of processing should happen in the sensor." Since event cameras already compute temporal contrast, much computing power can be saved in the overall system.

Additionally, when the same high temporal resolution (up to several hundred kHz) is required, standard cameras require much more power than event cameras. Such high-speed cameras typically also require massive external illumination.

1.1.3 Challenges

Since the data output of event cameras (a stream of events) is fundamentally different from that of standard cameras (a sequence of images), classical computer vision algorithms cannot be applied to event cameras, and so, a paradigm shift is needed. Event-driven algorithms that take into account the sparse and asynchronous nature of the data must be devised to unlock the advantages that event cameras offer.

From a processing point of view, the challenge consists of dealing with a different

[2] http://www.eetimes.com/document.asp?doc_id=1331675

represenation of the visual data: (*i*) sparse, asynchronous measurements (events) instead of dense, synchronous ones (frames), and (*ii*) brightness differences (i.e., temporal contrast) instead of absolute brightness.

The major difference is (*i*), due to a different sampling of the visual content of the scene: in standard cameras, sampling is based on an external clock that collects synchronous measurements and form a frame; in contrast, event cameras follow a data-driven sampling, i.e., based on brightness changes that happen in the scene relative to the motion of the sensor, and they happen sparsely on the image plane. Hence, event cameras do not conform to the implicit assumption of most computer-vision algorithms: the existence of frames.

Challenge (*ii*), which is based on the fact that each event only provides binary information (brightness increase or decrease, represented by the event polarity), is specific of some event cameras, such as the DVS. The ATIS [116], for example, reports absolute intensities. In the DVS, dealing with this binary information is challenging; however, it does not seem to be a problem, since it has been demonstrated that intensity gradients as well as absolute intensity can be reconstructed from the events [32, 66]. Moreover, depending on the application, intensity reconstruction may not be needed, and the binary representation provides sufficient information to design an algorithm that fulfills the task.

From a practical perspective, some further challenges need to be overcome. The noise levels of today's event cameras are still high and their resolution rather low (128×128 pixels for the DVS, 240×180 for the DAVIS). However, new sensor prototypes with higher resolution, more sensitivity, and color filters will overcome these limitations (see Chapter 3).

Since the events are asynchronous and the event rate depends on the camera motion, the scene, its texture, and the biases (camera parameters), guaranteeing real-time performance is challenging. Most algorithms have a constant computation time *per event* and, therefore, are "real-time" only up to a certain event rate.

1.1.4 Historic Development of Event Cameras

The first silicon retinas were developed in the early 90's by Misha Mahowald and Carver Mead [91, 90, 89]. Similar to the human retina, their silicon retina "reduces the bandwidth by subtracting average intensity levels from the image and reporting only spatial and temporal changes." [90] As noted by [36], "the performance of early systems suffered because they had to simultaneously combine a new computational paradigm with tricky delay-insensitive asynchronous logic and massively parallel analog computation." For more than a decade, only very little progress was reported, with the exception of [15]. Only in 2003, a simultaneous spatial contrast and local

orientation vision sensor was presented [128], showing that is was possible to overcome sensor mismatch and low performance by innovative circuit design and architecture. Finally in 2008, the first event camera became commercially available: the Dynamic Vision Sensor (DVS) [77] that was developed during the CAVIAR project [131] funded by the European Union. The project aimed at building an event-based hardware system consisting of sensing, processing, learning, and actuating using the Address-Event Representation (AER) communication framework. The system enables high-speed visual object recognition and tracking latencies in the order of milliseconds. The DVS is a 128×128 pixel array that responds to temporal brightness changes as described above and constitutes the sensing layer of the CAVIAR system.

Based on the DVS, several event cameras with additional functionalities have been developed. The Asynchronous Time-Based Image Sensor (ATIS) [116] combines a DVS with pulse-width modulated intensity encoding [119] for absolute brightness. Events by the DVS pixel trigger the readout of the absolute intensity that is encoded using the time interval between two additional events. The Dynamic and Active-pixel Vision Sensor (DAVIS) [19] combines a DVS with a standard camera, using the same pixel array. A more detailed review of silicon retinas is provided in [83, Chap. 3] and [8, Sec. 1.5].

1.2 State of the Art on Event Cameras

In this section, we describe the state of the art and related work of event cameras for robotics, discuss their strengths and weaknesses, and motivate the goals of this thesis. The section is divided in infrastructure, feature detection, feature tracking, ego-motion estimation, and robot control using event cameras.

1.2.1 Infrastructure for Event-based Vision for Robotics

Software Frameworks and Drivers. In 2006, the open-source software framework jAER[3] was initiated for acquiring, processing, and visualizing data in the Address-Event Representation. While this framework includes many drivers for neuromorphic sensors (such as event cameras and silicon cochleas), many filters, and advanced visualization methods, it is not suitable for embedded computers and high throughput.[4] Only very recently, a version optimized for embedded computers, cAER[5], and a standalone driver, libcaer[6], were released.

[3] https://sourceforge.net/projects/jaer/
[4] Mostly because it is written in Java.
[5] https://github.com/inilabs/caer
[6] https://github.com/inilabs/libcaer

In the robotics community, the Robot Operating System (ROS[7] [122]) became the *de facto* standard for robotic middleware in academia. While it provides many modules for popular sensors (laser scanners, cameras, GPS, IMU, etc.), no support was provided for event cameras. We developed an open-source interface[8] and basic visualization tools to make event cameras available on robotic platforms.

Intrinsic Calibration. For many computer-vision such as visual odometry or SLAM, intrinsic camera calibration is required. Many toolboxes exist and provide robust calibration routines for standard, frame-based cameras, typically using checkerboard calibration patterns. While the geometric lens models are the same for standard and event cameras, detection of the checkerboard is not trivial in the event stream. It was shown by [27] that LEDs blinking at high frequency can be easily detected by event cameras. We therefore developed a package that allows intrinsic (and also stereo) calibration of event cameras using a pattern of blinking LEDs [102] (see Fig. 2.1a). It is also possible to use a blinking pattern on a regular computer screen for calibration (see Fig. 2.1b).

Sensor Delay Characterization. To devise state estimation and control laws for a robot, sensor characterization is required. This includes the update rate and sensor latency. Due to the asynchronous nature of event cameras, the update rate is almost continuous and, therefore, we focused on the characterization of the latency. Similarly to [86], we measured the round-trip delay between triggering an LED and measuring that stimulus. Previous works measured the round-trip delay only on standard computers [34], while we also evaluated the latencies on embedded computers and compared it to popular frame-based cameras [97].

Simulators. Only one previous simulator for event cameras[9] is known to the author. It operates by thresholding the difference of the two latest images on a moderate framerate of around 60 Hz and is therefore unable to provide precise timestamps of the events. Furthermore, neither are events generated for smooth gradients nor are several events generated for strong gradients that cross the threshold multiple times.

We developed two simulators for different applications. The first is a sensor-in-the-loop simulation where a known video is shown to a intrinsically and extrinsically calibrated camera on a computer screen [102]. This has the advantage that realistic sensor noise is contained in the data. The second simulator is purely virtual and allows for arbitrary sensor sizes and parameters [103]. It is based on frames that

[7] http://www.ros.org/
[8] https://github.com/uzh-rpg/rpg_dvs_ros
[9] https://github.com/HBPNeurorobotics/gazebo_dvs_plugin

are rendered at high frequency and, therefore, the simulation is far from real time operation. However, it additionally allows to include depth measurements and other sensors into the simulation. We released this simulator as open-source package.[10]

Datasets. Computer-vision research has repeatedly benefitted from benchmark datasets. In [137], the benefits of having such a dataset for event cameras were highlighted. Only few and rather specialized datasets for event cameras existed: In [111], an approach to convert frame-based datasets to event-based representations. They released neuromorphic versions of the MNIST handwritten digit database as well as the Caltech101 object recognition dataset.[11] Their method uses an event camera on a pan-tilt unit and shows images on a computer screen.[12] Similarly in [61], conventional datasets for object tracking, action recognition, and object recognition have been converted to event streams.[13] Since the datasets they converted contain video sequences, the event camera could be held static as shown in Fig. 1.5. While these datasets allow direct comparison of event cameras with frame-based cameras, the datasets are based on sampled images. Since objects easily move more than one pixel per frame, these datasets cannot show the low-latency capabilities of event cameras or the benefit from the asynchronous nature of the event stream.

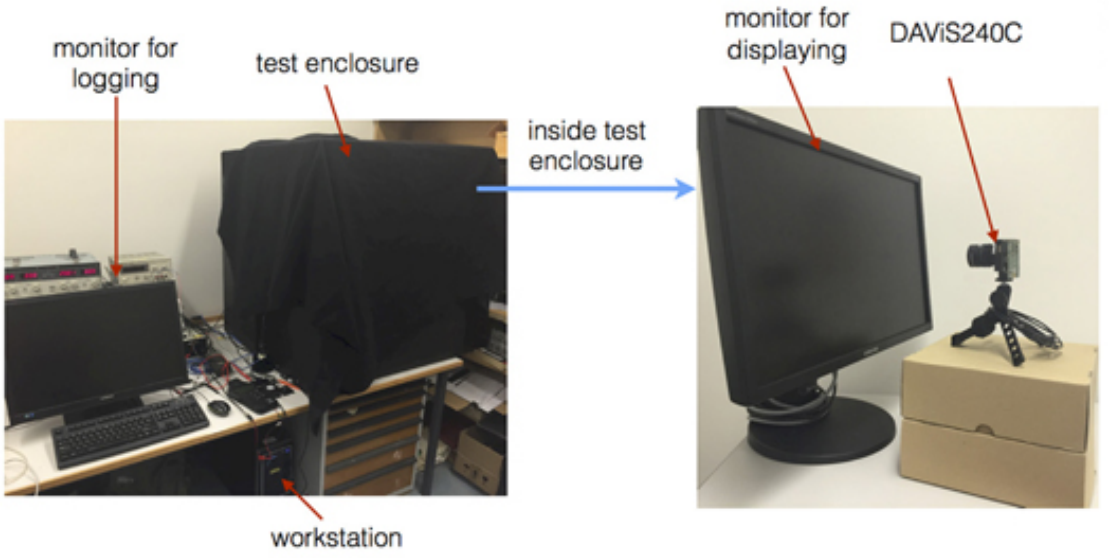


Figure 1.5: Converting conventional datasets to event streams by showing the frames on a computer screen to an event cameras [61]

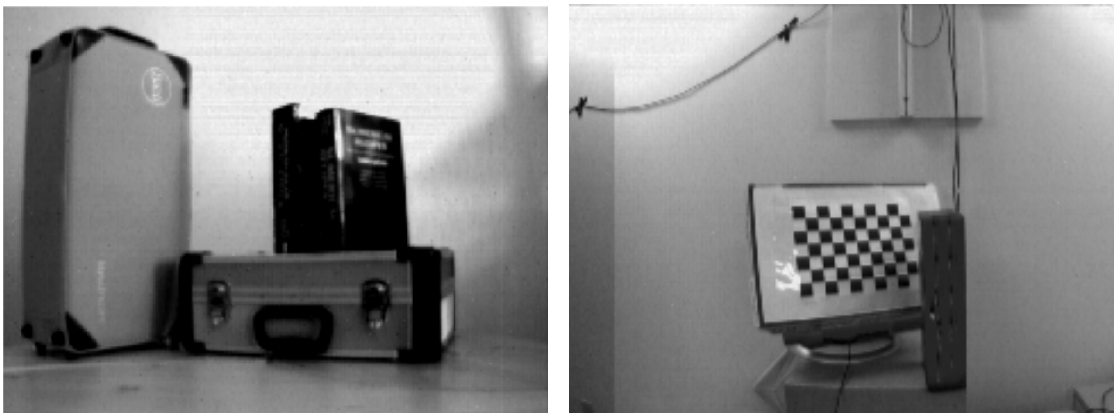
In [56], a dataset containing both egomotion and object motion recorded with a DVS is

[10] https://github.com/uzh-rpg/rpg_davis_simulator
[11] The datasets can be found on http://www.garrickorchard.com/datasets
[12] A video of the recordings can be found on https://youtu.be/2RBKNhxHvdw
[13] The datasets can be found on http://dgyblog.com/projects-term/dvs-dataset.html

presented.[14] For egomotion estimation, the DVS was mounted on a wheeled robot and, therefore, consists mostly of forward and backward motion in a plane. The dataset also contains sequences of single and multiple moving objects shown on a projector. Most scenes consist of simple shapes such as black-and-white circles. No ground truth for camera and object motion is provided.

In [127], a dataset for evaluation of optical flow was presented. Ground truth was computed using an inertial measurement unit.[15] They provide datasets with sharp gradients and low texture, such as checkerboards (see example in Fig. 1.6a).

Synthetic and real datasets of an event and depth camera (a DAVIS and a Microsoft Kinect) mounted on a mobile ground robot were released by [7].[16] The ground robot was equipped with a pan-tilt unit, resulting in 5 degrees of freedom. They provide datasets on static scenes with low texture (such as checkerboards) and large objects (see example in Fig. 1.6b).



(a) Optical-Flow Dataset [127] **(b)** Mobile Robot Dataset [7]

Figure 1.6: Related Datasets using Event Cameras

We created a dataset that contains handheld motion in a variety of scenarios and motion speeds, with accurate ground truth from a motion-capture system [103]. Unlike previous datasets, all six degrees of freedom are excited and contain natural textures. Example environments are shown in Fig. 2.2 Therefore, they are a realistic benchmark for robotic and virtual-reality applications.

[14]The dataset can be found here: http://www.itee.uq.edu.au/cis/projects#dvs
[15]The dataset can be found here: http://sensors.ini.uzh.ch/databases.html
[16]The dataset can be found here: https://github.com/fbarranco/eventVision-evbench

1.2.2 Event-based Feature Detection

Event Lifetime. Individual events contain very little information and are, therefore, highly ambiguous. Thus, any event-based algorithm relies on information from past events, or, sub-optimally, on the information contained in a collection of recent events. However, it is not trivial how to choose this set of previous events. Ideally, these past events form 1-pixel wide lines (i.e., edges) in the image plane (since 1 pixel is a natural measure of the minimum amount of motion of an edge in the image plane). Many works, however, use one of the two "naive" strategies to select the set of events for current processing: (*i*) using a fixed time interval (e.g. [130, 108]), (*ii*) using a fixed number of events (e.g. [69, 95, 55, 124]). Both strategies have severe shortcomings. The first strategy is motion-dependent: if the camera moves slowly, a larger time interval is required to show the same amount of events. The second strategy is scene-dependent: a larger number of past events is required for highly textured scenes. However, there exist situations where neither strategy can achieve the above-stated goal: if two objects in the scene are moving at different speeds, there are no parameters for either strategy that will allow sharp rendering of both objects. To overcome this limitation, we introduced the concept of event "lifetime" [99]. For each event, we compute its current velocity (apparent motion in the image plane) and use this as a prediction of the time that it takes for the moving edge to traverse 1 pixel. This allows to render sharp event images at any point in time.

Corner Detection. Features, such as corners, are a fundamental building block for many computer-vision applications such as visual odometry or object recognition. Examples for such features detectors are Harris [58] and FAST [126].

For event cameras, only two methods have been described in the literature. First, the method of [29] that works by fitting planes and searching for intersections. In [142], an event-based adaptation of the Harris was proposed. However, both methods are computationally expensive. We propose the first FAST-inspired method [96] that is an order of magnitude faster, while only performing slightly worse than previous methods.

1.2.3 Event-based Tracking

Lines and Circles. In this section, we review event-based tracking algorithms that estimate the motion from objects on the image plane. In [31], an event-based adaptation of the Hough transform was presented to track lines using only the events. A similar approach was presented in [107] that uses the Hough transform to detect circles. More recently, a line detector was presented in [18]. Event-based particle tracking was presented in [41], applied to high-speed tracking of particles in fluids.

In [102], we present the first method to track a set of lines for 6-DOF pose estimation for high-speed quadrotor maneuvers. In the experiments, we show that our algorithm can estimate the quadrotor pose in an event-by-event fashion, even during flips with rotational speeds of 1,200 °/s.

In [97], a ball was tracked using a stereo pair of DVS for evasive maneuvers with quadrotors. We use circular kernels similar to [72]. Event cameras allow ball detection, trajectory extrapolation, and collision prediction within 15 ms—less than half of the time between two frames of a standard 30 Hz camera.

Shapes. A method to track a predefined shape was presented in [108]. The approach uses an event-based adaptation of the Iterative Closest Point (ICP) algorithm. It is applied to high-speed stabilization of miniature gripper (see Fig. 1.7). Similarly in [72], predefined kernels were tracked using the event stream (see Fig. 1.8). However, instead of ICP, an infinite impulse response filter was used. In [141], the 3D pose of a model consisting of a set of lines was tracked using event-based updates that combine both 2D and 3D criteria.

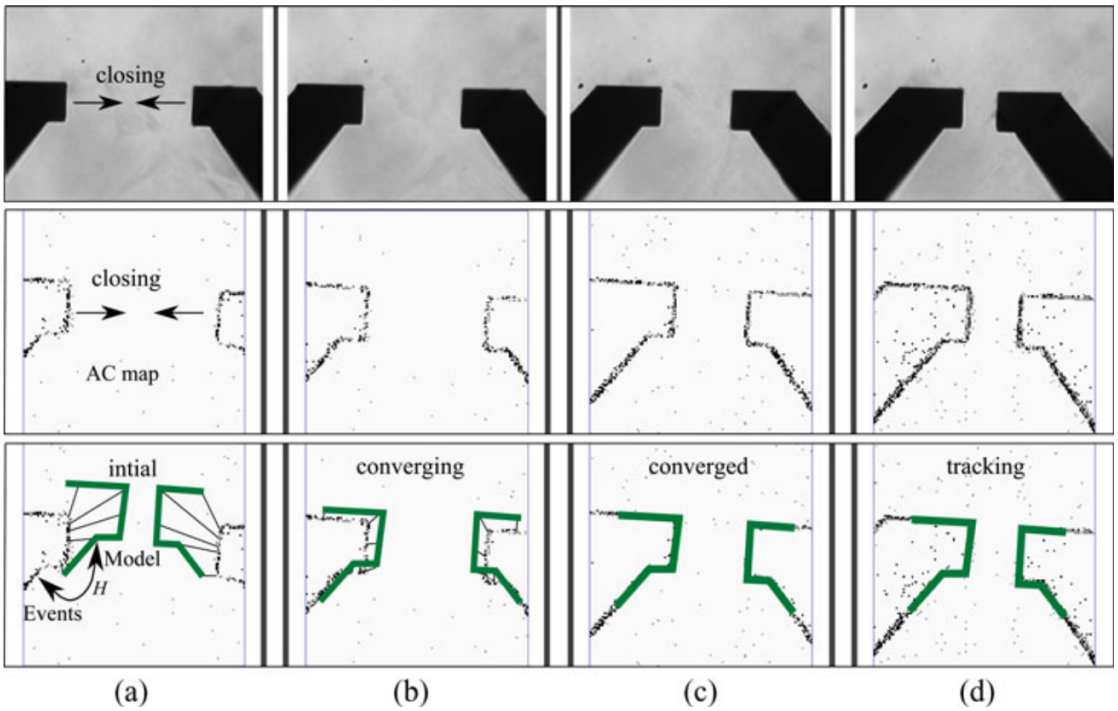
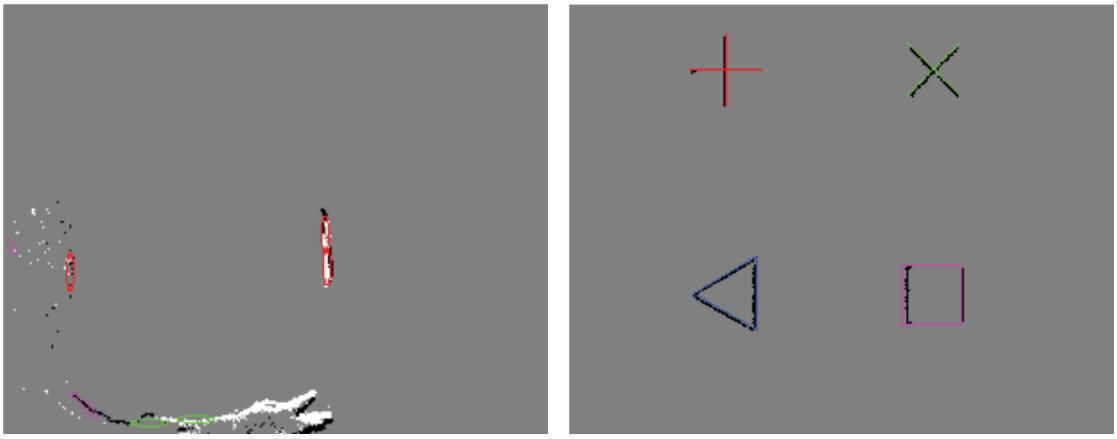


Figure 1.7: Event-based Iterative Closest Point Algorithm [108]. The top, middle, and bottom row shows a standard image, accumulated events, and the overlaid model, respectively. The model is shown as solid green lines. (a) shows the initial state and the correspondences as black thin lines, (b) the model is converging towards the real position, (c) convergence, and (d) the model keeps tracking the real position.



(a) Pen thrown in the Air. **(b)** Basic Shapes.

Figure 1.8: Event-based Kernel Tracking [72]

While [72] is restricted to manually-crafted kernels, we present a method [138] that can track arbitrary features. Using a DAVIS that provides both events and frames, we initialize features from the frames and track them using the events using event-based ICP. This allows feature tracking on natural textures that was not possible with previous methods.

Optical Flow. Several methods to compute optical flow with event cameras have been presented. In [10], an event-based adaptation of the classical frame-based Lukas-Kanade method [85] was devised. Estimation using plane fitting on a map with the timestamp of the latest event per pixel was used in [9]. A comparison of several state-of-the-art optical flow methods was presented in [127], along with some datasets used for the comparison. The comparison included a direction selective filter [33], four variants of a basic Lucas-Kanade algorithm [10] and four variants of local plane fits [9].

Optical flow has been estimated along with image gradient (the spatial derivative that triggered the events), jointly, in [32] for rotational motion, and in [6] for arbitrary dynamic scenes. In [32], a distributed message-passing algorithm consisting of local gradient descent operations is used that jointly estimated image gradient, angular velocity, and optical flow. The problem is posed in a variational optimisation framework in [6], with a data term consisting of the event stream and with spatial and temporal regularizers over a sliding-window time interval. Optimisation was carried out using a primal-dual algorithm, which required a GPU implementation to run in near real-time.

1.2.4 Event-based Motion Estimation

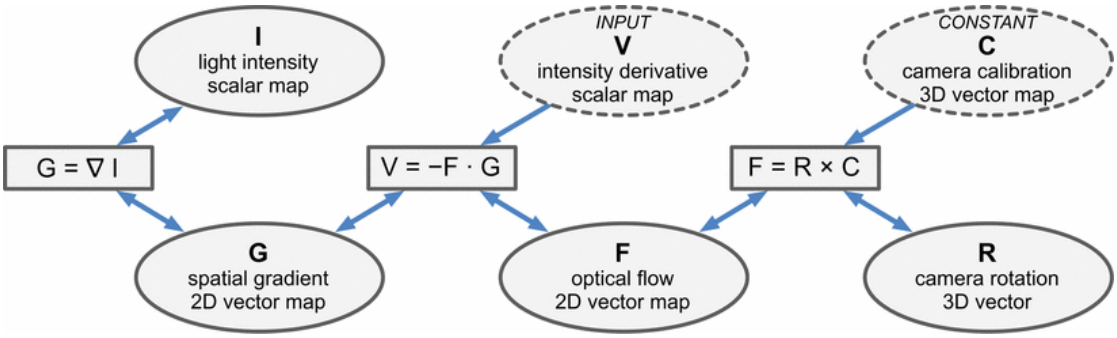
In this section, we review approaches that estimate the ego-motion of an event camera. Many of them either require additional sensing or restrict the degrees of freedom, the types of scenes, or both.

Rotation only. In [32], interacting maps were used to estimate the rotation of an event camera, while at the same time reconstructing an intensity image. Interacting maps are a network of recurrently interconnected areas that encode different aspects of visual interpretation (e.g., optic flow, intensity image, etc.) as shown in Fig. 1.9a. Because each area tries to be consistent with its neighboring areas, they converge towards a global mutual consistency (see Fig. 1.9b). As information can flow in both ways, this breaks with the traditional feed-forward processing scheme. In [66], similar output quantities (G, I, and R) were produced using a pair of Bayesian filters operating in parallel (see Fig. 1.10a): a particle filter for to track the rotational motion of the camera, and a per-pixel Extended Kalman Filter (EKF) to compute the image gradient (G) from which the intensity image (I) was obtained by Poisson integration. The method of [55] does not require to compute the intensity image to estimate the camera motion and works by maximizing the contrast of an image of motion-warped events.

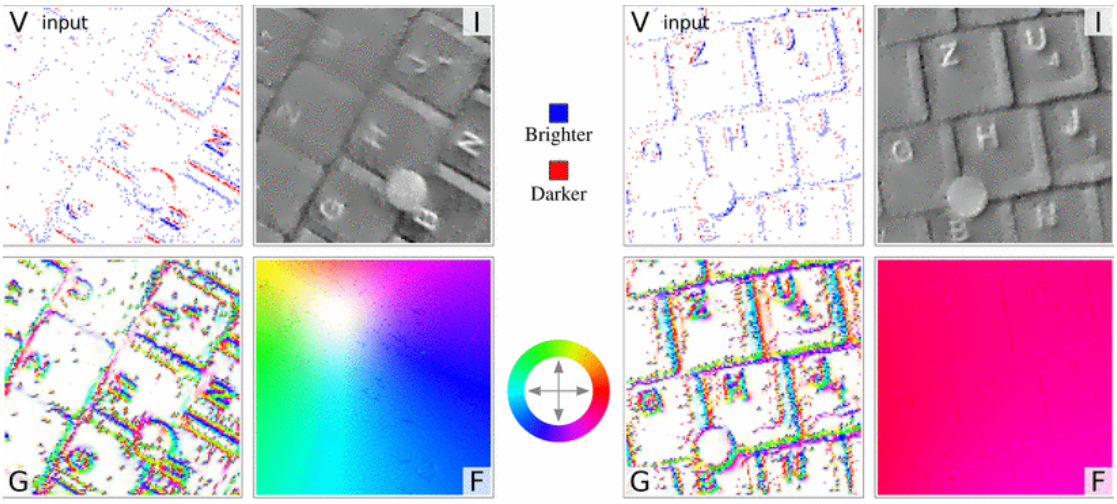
2D Estimation. Localization in 2D was presented by [145] and later extended to Simultaneous Localization and Mapping (SLAM) [146] (see Fig. 1.10b) and exploration [60]. The system design, however, was limited to slow planar motions (i.e., 3 DOF) and planar scenes parallel to the plane of motion consisting of high-contrast black-and-white line patterns. They used an upward-looking event camera on a ground robot and created gradient maps from the ceiling using probabilistic updates.

Additional Sensing. In [26], a visual-odometry algorithm is presented that uses an event camera in combination with a frame-based camera. They track the camera motion between two frames using the events by incorporating a probabilistic sensor model. In [144], a 3D SLAM system was presented that uses an event camera with an additional depth sensor (see Fig. 1.11). Localization in predefined line-based maps was presented in [149]. They use an inertial measurement unit to align the events with gravity and propose a fast algorithm to detect vertical lines.

6-DOF SLAM. Only recently, 6-DOF SLAM algorithms were presented. In [67], three decoupled probabilistic filters were used to estimate 6-DOF camera motion, the intensity gradient image, and an inverse depth map (see Fig. 1.12). At the same time, the intensity gradient image is used to reconstruct a video signal. The method is able to



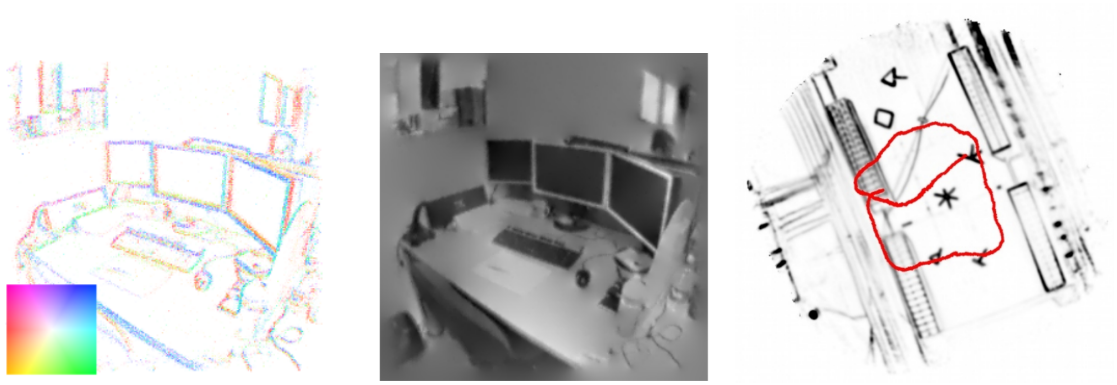
(a) Figure from [32]. Network architecture, showing the "maps" (represented by ellipsoids) and the relationships (i.e., per-pixel equations) between them (represented by rectangles). The distributed algorithm jointly estimates the image gradient (G), absolute intensity (I), optical flow (F), and angular velocity (R) from the input event stream (V).



(b) Results of [32]. The map V shows the events from the event camera (input to the algorithm) as it moves above part of a keyboard. Red and blue pixels represent negative and positive events, respecetively. All other maps (image I, gradient G and optical flow F) are inferred from V. The hue in G and F represents the direction of the vector at the given pixel (cf. the legend at the center), and the saturation is proportional to the vector norm. Left: shows the result of a clockwise rotation of the input around the image center. Right: shows the result of a left to right rotational motion of the input.

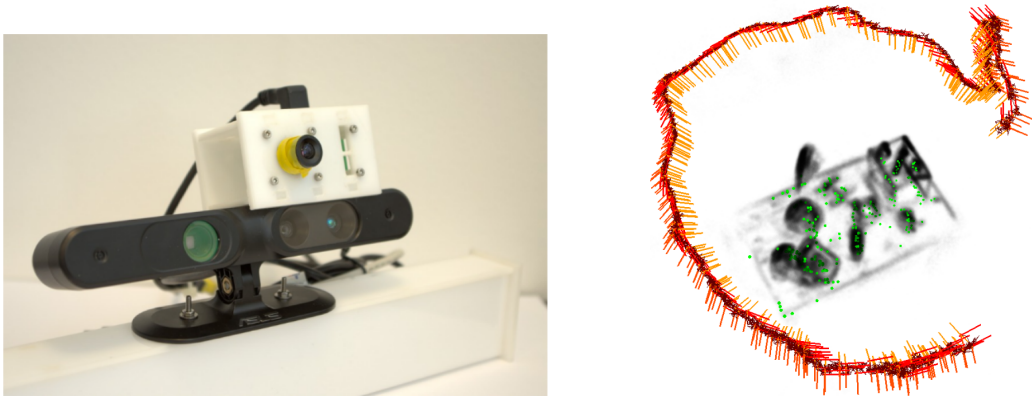
Figure 1.9: Interacting Maps for Fast Visual Interpretation [32]

track a hand-held event camera in unknown and unstructured environments. However, the method is computationally intense and requires a GPU for computation. In [124], a geometric approach to 6-DOF parallel tracking and mapping was presented. The method combines an image-to-model alignment by accumulating events to create an artificial image and an event-based 3D reconstruction algorithm [123]. In contrast to [67], the method does not require the reconstruction of the gradient image and can produce several hundred pose estimates per second on a standard CPU. An example of a computed map and trajectory is shown in Fig. 1.13.



(a) Mosaicing and Tracking [66] **(b)** 2D SLAM [146]

Figure 1.10: Event-based Motion Estimation



(a) DVS (top) with depth camera (below) **(b)** Map and camera trajectory

Figure 1.11: Event-based 3D SLAM with an Depth-augmented Event Camera [144].

Sparse Event-Based Visual Odometry. Prior to [67] and [124], we developed a sparse, feature-based 6-DOF visual-odometry algorithm [70] that was based on low-latency feature tracks [138]. Our odometry algorithm worked on natural scenes and used arbitrary features, with arbitrary shapes, defined by the edgemaps in the frames of the DAVIS and tracked using the event stream. These feature tracks were fed to an event-based visual-odometry algorithm that interleaved robust pose estimation and probabilistic mapping.

Probabilistic Dense Tracking. In [54], we presented a probabilistic filter framework to track the event-camera pose in a known map. This was the first algorithm that allowed for 6-DOF event-camera tracking in natural environments. The method provided pose updates upon the arrival of each event and thus virtually eliminated latency.

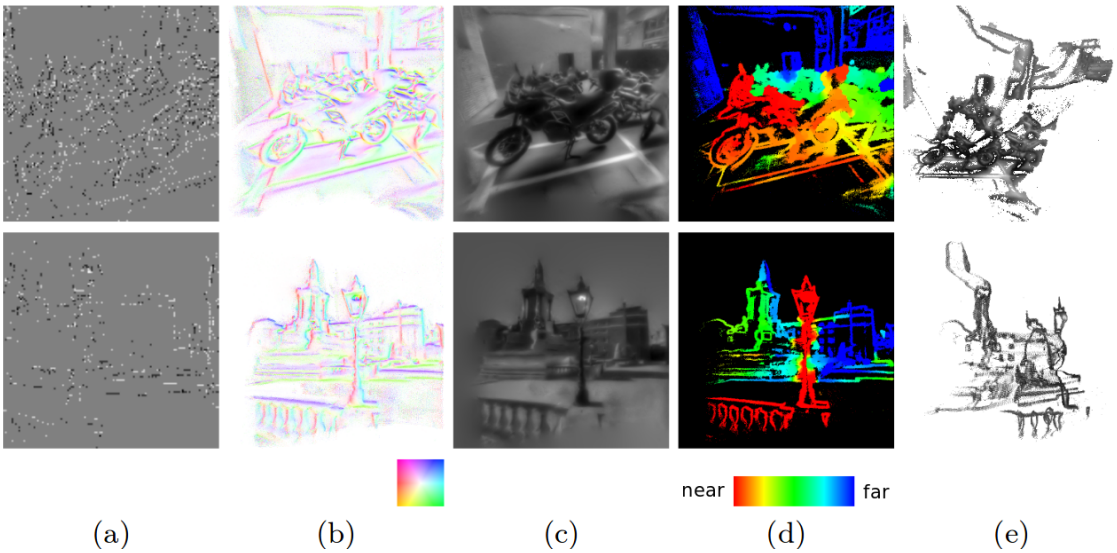


Figure 1.12: 3D Reconstruction and 6-DoF Tracking [67]. (a) event visualization, (b) gradient estimation, (c) intensity reconstruction, (d) depth estimation, and (e) semi-dense point cloud.

Figure 1.13: EVO [124]: geometric approach to 6-DOF semi-dense parallel tracking and mapping in real time on a CPU.

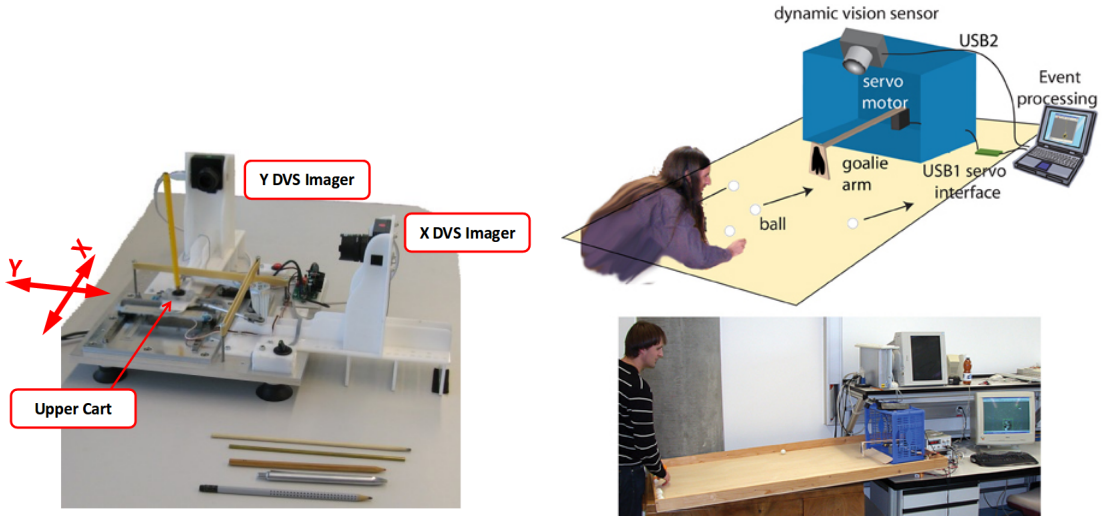
Continuous-Time Visual-Inertial Trajectory Estimation. Finally, in [101, 100], we proposed to use a continuous-time trajectory estimation framework. This framework allows a direct and natural integration of all events and inertial measurements in a single optimization problem, while using only few optimization parameters.

1.2.5 Event-based Motion Control

The first closed-loop control application using event cameras was presented in [31], where a pair of DVS was used to balance a pencil on a highly-reactive platform moving

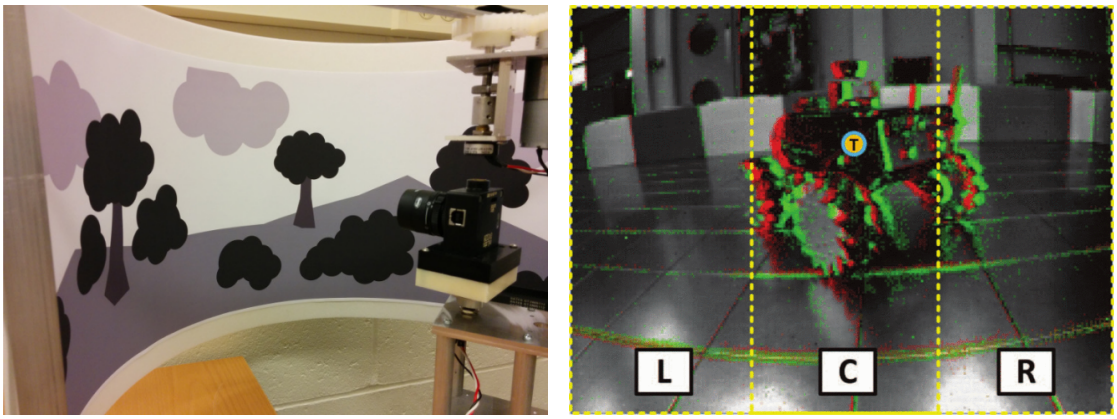
in a plane.[17] The setup is depicted in Fig. 1.14a. The key was an event-based adaptation of the Hough transform.

In [34], a robotic goalie was presented that defended incoming balls with very low latency.[18] The setup is illustrated in Fig. 1.14b.



(a) Pencil Balancing [31]

(b) Robotic Goalie [34]



(c) Heading Tracking [104]

(d) Predator Robot [95]

Figure 1.14: Examples of Event-based Control Applications

In [104], an event-based control algorithm to stabilize yaw of a 1-DOF robot (see Fig. 1.14c) is presented. Their computationally efficient algorithm performs an approximation to Bayesian inference that allows event-by-event updates. They show that an event camera outperforms standard cameras with respect to computational load, data rate, bandwidth, and latency in this task.

[17] Video of pen balancing: https://youtu.be/f9UngTdngY4
[18] Video of robotic goalie: https://youtu.be/PB5vWvT_QAA

These three works demonstrated the low-latency capabilities of event cameras in an impressive way. Furthermore, the also showed that the algorithms needed are very efficient: the pen-balancing algorithm was running on an embedded microcontroller, while the robotic goalie was only using 4 % CPU of a standard desktop. This comes from the sparse, asynchronous output of event cameras that only reports changes.

In [95], a Convolutional Neural Network (CNN) for a predator robot is trained using both the events and frames from a DAVIS. The method accumulates a fixed number of events, thus the sampling rate depends on the speed of the robot. The network generates four outputs (left, middle, right, and not visible; see Fig. 1.14d) that are used as steering commands and achieves accuracies of around 90 %.[19]

In [37], we present a method for high-frequency tracking and control of slot cars, enabling human-vs-machine races.[20] The event camera was placed in "eye-of-god" view. Throttle and brake commands are computed at 666 Hz, while requiring less than 3 % of CPU load. An image of a human-vs-machine race is shown in Fig. 2.11.

1.3 Summary

In this chapter, we have discussed the working principle of event cameras and given a historical perspective of their development. We have also mentioned the advantages of the event cameras over standard cameras (low latency, high speed, high dynamic range, low power, etc.) and the challenges that the bio-inspired data-driven sampling scheme poses on developing new computer vision algorithms to unlock the potential of event cameras.

In the second half of the chapter, we have discussed the state of the art on the five topics developed in this thesis: infrastructure, feature detection, feature tracking, ego-motion esimation, and robot control using event cameras. During the literature review, we have pointed the most relevant works on the above-mentioned topics and we have put in context several of our contributions, which we will further detail in the next chapter.

[19] Video of predator robot: https://youtu.be/lPF3Youpmqk
[20] Video of slot-car racing: https://youtu.be/CnGPGiZuFRI

2 Contributions

This chapter summarizes the key contributions of the papers that are reprinted in the appendix. It further highlights the connections between the individual results and refers to related work and video contributions. Because Papers A and B contain two main contributions, they are mentioned in both sections where they are relevant (Sections 2.1 and 2.3). In total, this research has been published in eight peer-reviewed conference publications and one journal publication in the *International Journal of Robotics Research (IJRR)*. Two further journal papers are currently under review at the *IEEE Transactions on Robotics (TRO)* and the *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*. These works led to a *Qualcomm Innovation Fellowship* in 2016 and were awarded the *Misha Mahowald Prize for Neuromorphic Engineering* in 2017.

2.1 Infrastructure for Event-based Vision

Since event cameras are relatively new, few publicly available tools exist. The community was mostly relying on a Java-based framework (jAER)[1], which is prohibitive for small processors as those used on MAVs. We thus first developed a ROS interface for the DVS [77] and DAVIS [19] event cameras, designed and implemented an intrinsic camera calibration tool, and characterized the latencies. Further, we released the first event-camera dataset and simulator for tracking and visual odometry. This dataset allows researchers without access to expensive hardware to develop algorithms and test them on real data, and serves as benchmark to compare their performance against existing algorithms. These contributions are summarized in more detail below.

[1] https://sourceforge.net/projects/jaer/

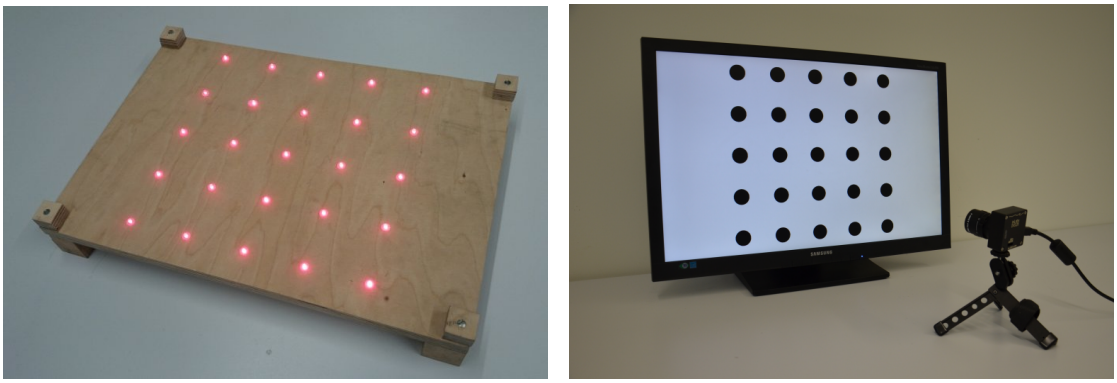
2.1.1 Paper A1: Event Camera Driver and Calibration

(P1) E. Mueggler, B. Huber, and D. Scaramuzza. "Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2014, pp. 2761–2768. DOI: 10.1109/IROS.2014.6942940

We developed a driver for the DVS and DAVIS event cameras that allows their direct integration in the Robot Operating System (ROS) framework. Because for many computer-vision tasks the intrinsic camera calibration is needed, we also developed a calibration toolbox. Since traditional methods do not work directly with events, we used active patterns that blinked at high frequency to generate events. We used both an LED pattern at 1 kHz and standard computer screens, whose backlight is typically dimmed with pulse-width modulation at several hundred Hertz. We prefer the first method as the maximum viewing angle, and thus the calibration accuracy, are typically higher (60° vs 20°). For the DAVIS, which provides image frames from the same physical pixels as the event sensor, the frames can be easily used for accurate intrinsic camera calibration using standard methods. The driver also provides support for hardware-synchronized stereo setups. We released both the driver and the calibration tools as open source.

Related Software

(S1) https://github.com/uzh-rpg/rpg_dvs_ros



(a) Array of Blinking LEDs **(b)** Dimmed Computer Screen

Figure 2.1: Calibration of Event Cameras. (a): Board with blinking LEDs for intrinsic and extrinsic calibration of the DVS stereo setup. The LEDs are blinking at a frequencies of 1 kHz, such that they can easily be detected by a DVS. (b): Alternatively, a dimmed computer screen can be used to show a calibration pattern, since dimming is typically implemented using pulse-width modulation at high frequency that is detectable by an event camera.

2.1.2 Paper B1: Event Camera Delay Characterization

(P2) E. Mueggler, N. Baumli, F. Fontana, and D. Scaramuzza. "Towards Evasive Maneuvers with Quadrotors using Dynamic Vision Sensors". In: *Eur. Conf. Mobile Robots (ECMR)*. 2015, pp. 1–8. DOI: 10.1109/ECMR.2015.7324048

To use event cameras for closed-loop, low-latency control (such as evasive maneuvers with quadrotors [97]), it is crucial to have precise estimates of the sensor latency. While the latency of the events is in the order of microseconds on the DVS, it is typically connected via USB to a host computer or robot, which introduces significant latency. In this work, we evaluated the round-trip delay from both a stimulus (e.g., an LED) to its detection in the event stream. We measured this round-trip delay for both a laptop and embedded computer for the DVS, a machine-vision camera, and the Kinect-like ASUS Xtion. We measured delay below 3 ms and 5 ms for a laptop and embedded computer, respectively, which is below the typical exposure time of standard cameras. In addition, since event cameras operate asynchronously, a changing stimulus is detected immediately (without the delay of $1/f$ seconds, where f is the sensor frequency in Hz). These results suggest that, for most practical robotic applications, a standard USB interface should be sufficiently fast.

2.1.3 Paper C: Event Camera Dataset and Simulator

(P3) E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza. "The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM". in: *Int. J. Robot. Research* 36 (2 2017), pp. 142–149. DOI: 10.1177/0278364917691115

Publicly available datasets have a long and successful history in the domain of computer vision, machine learning, and robotics. They serve as benchmark to compare different algorithms and make the data available to researchers that lack expensive experimental infrastructure. In [137], the authors address the challenges and state-of-the-art for neuromorphic vision datasets. They finally remark that "dataset creation should be prioritized and recognized as a task of utmost importance to the field." Existing datasets are either not suited for egomotion estimation [111, 61] or do not excitate all six degrees of freedom [127, 7]. We present and release the first collection of datasets captured with a DAVIS in a variety of synthetic and real environments, which we hope will motivate research on new algorithms for high-speed and high-dynamic-range robotics and computer-vision applications. In addition to global-shutter intensity images and asynchronous events, we provide inertial measurements and ground-truth camera poses from a motion-capture system. The latter allows comparing the pose accuracy of ego-motion estimation algorithms quantitatively. All the data are released both as standard text files and as binary files (i.e., rosbag). We additionally release the first event-camera simulator as open source to create synthetic event-camera data.

Related Software

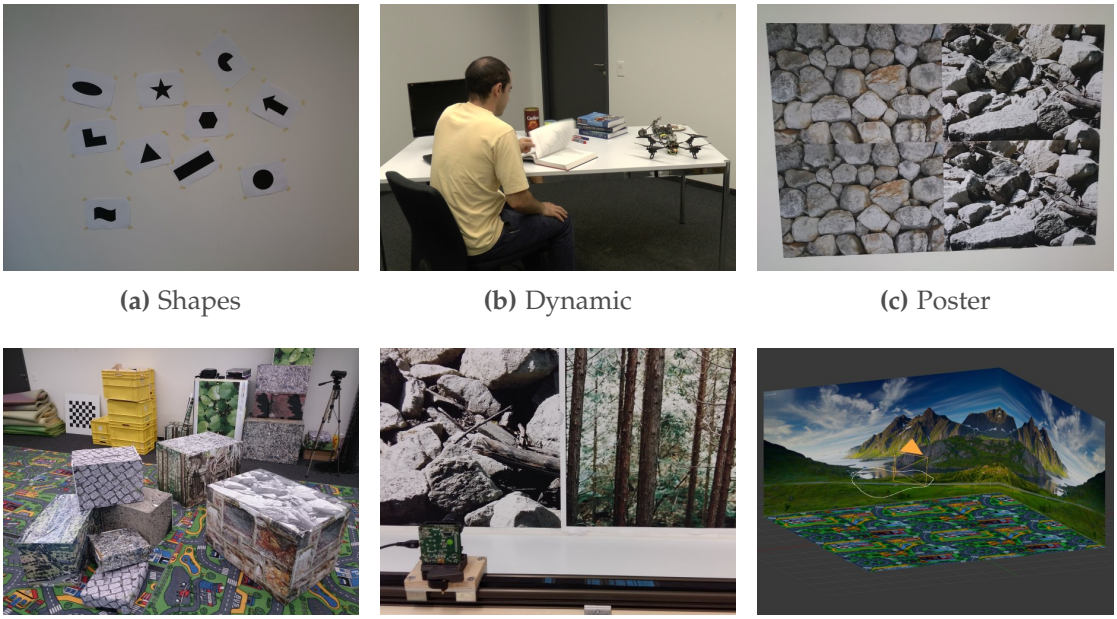
(S2) https://github.com/uzh-rpg/rpg_dvs_simulator

Related Datasets

(D1) http://rpg.ifi.uzh.ch/davis_data.html

Related Videos

(V1) https://youtu.be/bVVBTQ7l36I



(a) Shapes **(b)** Dynamic **(c)** Poster

(d) Boxes **(e)** Linear Slider **(f)** Simulator

Figure 2.2: Example Scenes from the Event-Camera Dataset [103]

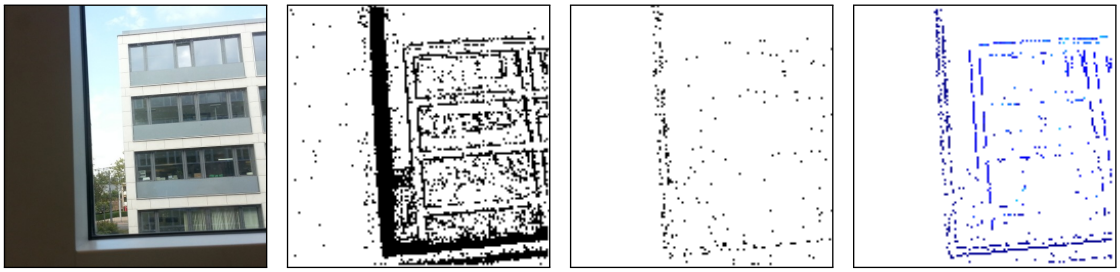
2.2 Event-based Feature Detection

In this section, the contributions for low-level feature detection and tracking are described. Such methods act as building blocks for higher-level algorithms, such as ego-motion estimation, which will be presented in the next section.

2.2.1 Paper D: Lifetime of Events

(P4) E. Mueggler, C. Forster, N. Baumli, G. Gallego, and D. Scaramuzza. "Lifetime Estimation of Events from Dynamic Vision Sensors". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2015, pp. 4874–4881. DOI: 10.1109/ICRA.2015.7139876

While an event is precisely timestamped upon its generation, it does not contain any information about the duration it is a valid measurement of the scene, i.e., it does not encode knowledge for how long the gradient causing the event remains at that particular pixel location. To overcome this issue, we introduce the concept of event *lifetime*, which can be summarized as the time that it takes for the moving edge causing the event to traverse the distance of 1 pixel, and present an algorithm to compute it from the event's velocity on the image plane. The lifetime endows the events with a finite temporal extent for a proper continuous representation of events in time. A direct application of this augmented stream (events plus lifetime) is the construction of sharp gradient (edge-like) images at any time instant, providing a solution to the tradeoff between completeness and motion blur of integrated event images.



(a) Image of the scene **(b)** 30 ms **(c)** 1 ms **(d)** Our method

Figure 2.3: An event camera is moved in front of a window frame diagonally, from bottom-left to top-right (a). Since the window frame is much closer than the buildings, its apparent motion is significantly larger. Thus, if we use a fixed accumulation interval, the images will either be blurred, if the interval is too long (b), or some structures will not be clearly visible, if the interval is too short (c). Our method estimates the lifetime of each event independently and displays the event for that period of time (d).

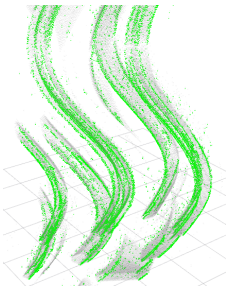
2.2.2 Paper E: Event-based Feature Detection

(P5) E. Mueggler, C. Bartolozzi, and D. Scaramuzza. "Fast Event-based Corner Detection". In: *British Machine Vis. Conf. (BMVC)*. 2017

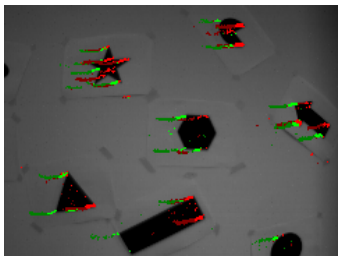
In textured scenes with rapid motion, event cameras output millions of events per second. Therefore, state-of-the-art event-based algorithms either require massive parallel computation (e.g., a GPU) or depart from the event-based processing paradigm. Inspired by frame-based pre-processing techniques that reduce an image to a set of features, which are typically the input to higher-level algorithms, we propose a method to reduce an event stream to a *corner event* stream. Our goal is twofold: extract relevant tracking information (corners do not suffer from the aperture problem) and decrease the event rate for later processing stages. Unlike previous works that required convolutions and matrix multiplications (Harris-like detector [142]) or plane fitting [29], we present the first method that is inspired by FAST [126]. Our event-based corner detector is very efficient due to its design principle, which consists of working on the Surface of Active Events (a map with the timestamp of the latest event at each pixel) using only comparison operations. Our method asynchronously processes event by event with very low latency. Our implementation is capable of processing millions of events per second on a single core (less than a *micro*-second per event) and reduces the event rate by a factor of 10 to 20.

Related Videos

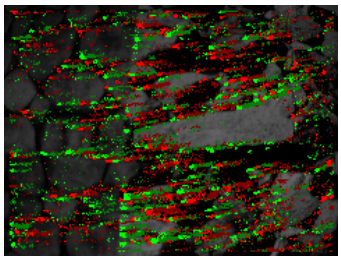
(V2) https://youtu.be/tgvM4ELesgI



(a) Raw Event vs Corner Event Stream



(b) Corners



(c) Corners

Figure 2.4: Event-based Corner Detection. (a): The output of our method is a corner event stream (green), which is here overlaid on the raw event stream (gray) in space-time (time going upwards). (b, c): Detected corners in the event stream (color indicates polarity) overlaid on intensity image (only used for visualization purposes).

2.3 Event-based Feature Tracking

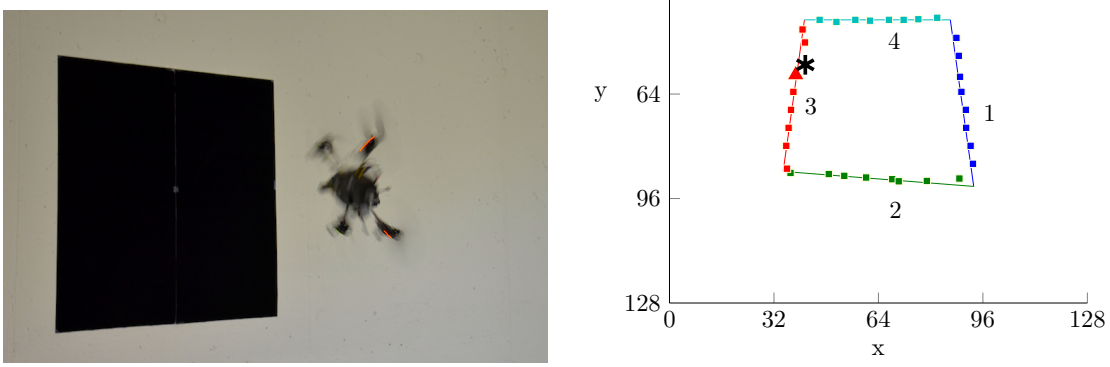
2.3.1 Paper A2: Tracking Polygonal Shapes

(P1) E. Mueggler, B. Huber, and D. Scaramuzza. "Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2014, pp. 2761–2768. DOI: 10.1109/IROS.2014.6942940

We developed the first 6-DOF pose tracking method using an event camera without any additional sensing. Our method is able to track a quadrotor's position and orientation with respect to a known pattern using an event camera *onboard* the vehicle. The method is able to track the pattern even under high-speed motion, such as quadrotor flips where rotational speeds of up to 1,200 °/s are achieved. Our algorithm works in an event-based fashion, such that the pose is updated asynchronously upon the arrival of a new event. More specifically, the algorithm minimizes the point-to-line reprojection error of the events using a model consisting of a set of line segments. Therefore, the method is limited to black-and-white scenes consisting of straight edges. We show that despite the low spatial resolution of the DVS, the pose can be tracked accurately.

Related Videos

(V3) https://youtu.be/LauQ6LWTkxM



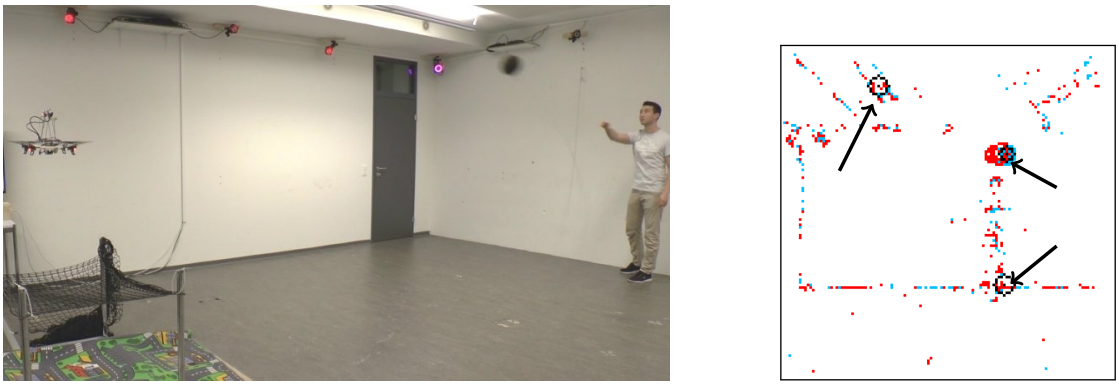
(a) Quadrotor Flip. **(b)** Tracking Algorithm.

Figure 2.5: Polygonal Shape Tracking.

2.3.2 Paper B2: Ball Tracking

(P2) E. Mueggler, N. Baumli, F. Fontana, and D. Scaramuzza. "Towards Evasive Maneuvers with Quadrotors using Dynamic Vision Sensors". In: *Eur. Conf. Mobile Robots (ECMR)*. 2015, pp. 1–8. DOI: 10.1109/ECMR.2015.7324048

In this paper, we estimate the trajectory of a ball thrown towards a quadrotor using a pair of event cameras in a stereo configuration. The aim of this project is to predict a collision of the ball with the quadrotor and, if necessary, perform an evasive maneuver. Our method tracks spherical objects on the image plane using probabilistic trackers that are updated with each incoming event. The object's trajectory is estimated using an Extended Kalman Filter with a mixed state space that allows incorporation of both the object's dynamics and the measurement noise in the image plane. Using error-propagation techniques, we predict a collision if the 3σ-ellipsoid along the predicted trajectory intersects with a safety sphere around the quadrotor. We experimentally demonstrate that our method allows initiating evasive maneuvers early enough to avoid collisions.



(a) Experimental Setup. **(b)** Ball Tracking.

Figure 2.6: Ball Tracking for Evasive Maneuvers. (*a*): ball (black) thrown towards a quadrotor. A leash was attached to the ball to prevent a potential collision. (*b*): Red and blue points indicate events with positive and negative polarity, respectively. The active circle trackers are marked in black and highlighted with an arrow.

2.3.3 Paper F: Event-based Feature Tracking

(P6) D. Tedaldi, G. Gallego, E. Mueggler, and D. Scaramuzza. "Feature Detection and Tracking with the Dynamic and Active-pixel Vision Sensor (DAVIS)". in: *Int. Conf. Event-Based Control, Comm. Signal Proc. (EBCCSP)*. Krakow, Poland, June 2016, pp. 1–7. DOI: 10.1109/EBCCSP.2016.7605086

We present the first algorithm to detect and track visual features using both the frames and the event data provided by the DAVIS. Features are first detected in the grayscale frames and then tracked asynchronously in the blind time between frames using the stream of events. To best take into account the hybrid characteristics of the DAVIS, features are built based on large, spatial contrast variations (i.e., visual edges), which are the source of most of the events generated by the sensor. An event-based algorithm is further presented to track the features using an iterative, geometric registration approach. The performance of the proposed method is evaluated on real data acquired by the DAVIS.

Related Videos

(V4) https://youtu.be/nglfEkiK308

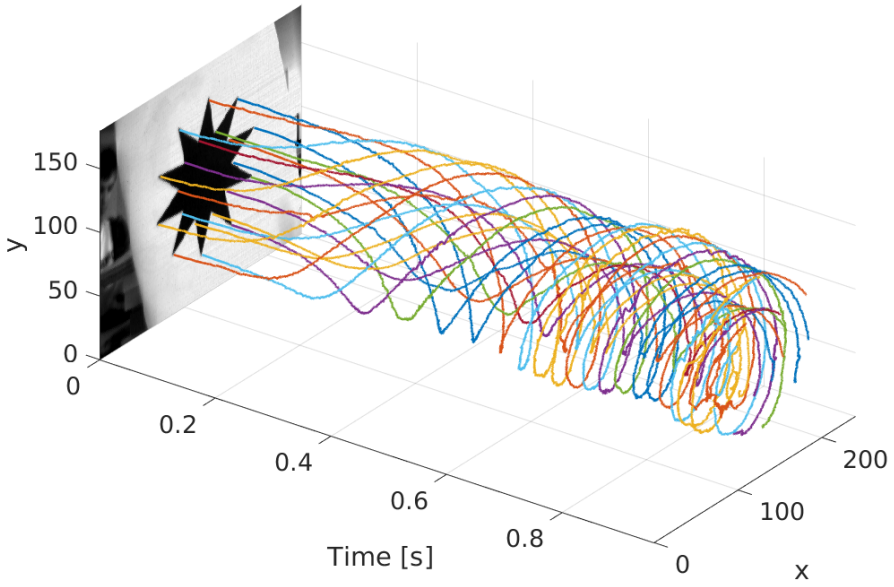


Figure 2.7: Feature Tracking.

2.4 Event-based Ego-Motion Estimation

In this section, we describe algorithms for ego-motion estimation using event-cameras, that is, algorithms that estimate the pose of an event camera with respect to a known map of the environment.

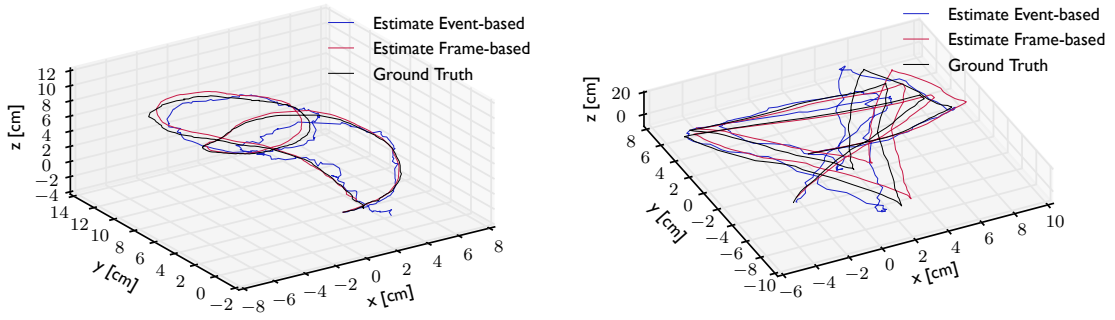
2.4.1 Paper G: Sparse Visual Odometry

(P7) B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza. "Low-latency Visual Odometry using Event-based Feature Tracks". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. Daejeon, Korea, Oct. 2016, pp. 16–23. DOI: 10.1109/IROS.2016.7758089

We present the first sparse, feature-based visual-odometry algorithm using the events and frames from the DAVIS sensor. Features are first detected in the grayscale frames and then tracked asynchronously using the stream of events, as described in Paper F. The features are then fed to an event-based visual odometry algorithm that tightly interleaves robust pose optimization and probabilistic mapping. We show that our method successfully tracks the 6-DOF motion of the sensor in natural scenes.

Related Videos

(V5) https://youtu.be/RDu5eldW8i8



(a) Office Spiral Dataset. **(b)** Office Zig-Zag Dataset.

Figure 2.8: Sparse Visual Odometry using Event-based Feature Tracks.

2.4.2 Paper H: Dense 6-DOF Tracking

(P8) G. Gallego, J. E. A. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza. "Event-based, 6-DOF Camera Tracking for High-Speed Applications". In: *IEEE Trans. Pattern Anal. Machine Intell.* (2017). under review

We present an event-based approach for ego-motion estimation, which provides pose updates upon the arrival of each event, thus virtually eliminating latency. Our method is the first work addressing and demonstrating event-based pose tracking in six degrees-of-freedom (DOF) motions in realistic and natural scenes, and it is able to track high-speed motions. The method is successfully evaluated in both indoor and outdoor scenes with significant depth variation, and under motions with excitations in all 6-DOFs.

Related Videos

(V6) https://youtu.be/iZZ77F-hwzs

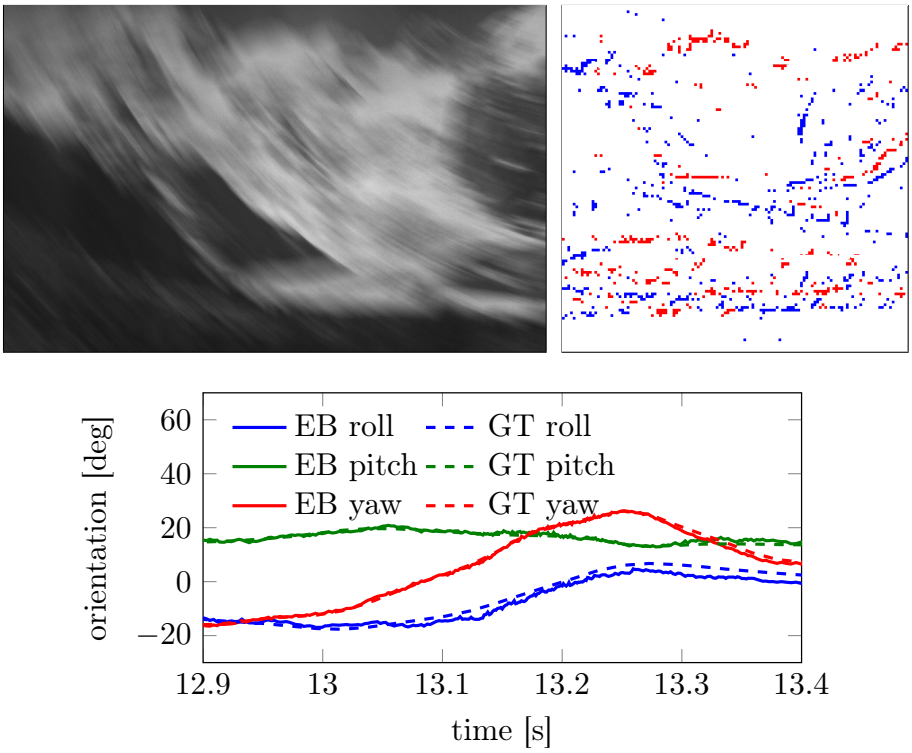


Figure 2.9: High-speed motion sequence. Top left: image from a standard camera, suffering from blur due to high-speed motion. Top right: set of asynchronous DVS events in an interval of 3 milliseconds, colored according to polarity. Bottom: estimated poses using our event-based (EB) approach, which provides low latency and high temporal resolution updates. Ground truth (GT) poses are also displayed.

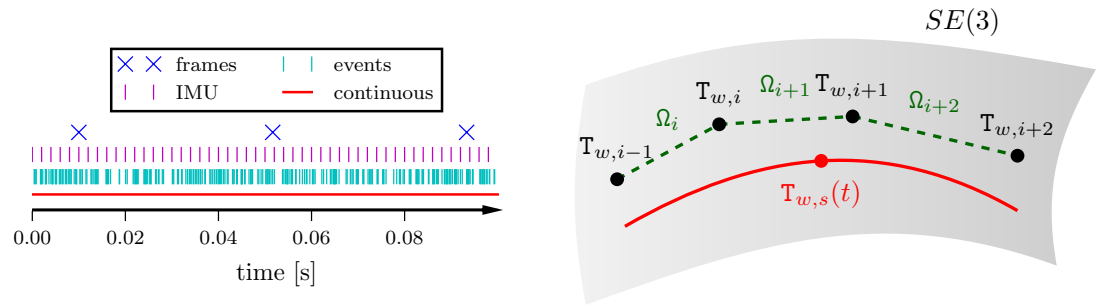
2.4.3 Paper I: Continuous-Time Trajectory Estimation

(P9) E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza. "Continuous-Time Visual-Inertial Trajectory Estimation with Event Cameras". In: *IEEE Trans. Robot.* (2017). under review

In this paper, we leverage a continuous-time framework to perform trajectory estimation by fusing visual data from a moving event camera with inertial data from an IMU. This framework allows direct integration of the asynchronous events with micro-second accuracy and the inertial measurements at high frequency. The pose trajectory is approximated by a smooth curve in the space of rigid-body motions using cubic splines. This formulation significantly reduces the number of variables in trajectory estimation problems. We evaluate our method on real data from several scenes and compare the results against ground truth from a motion-capture system. We show superior performance of the proposed technique compared to non-batch event-based algorithms. We also show that both the map orientation and scale can be recovered accurately by fusing events and inertial data. To the best of our knowledge, this is the first work on visual-inertial fusion with event cameras using a continuous-time framework.

Related Publications

(R1) E. Mueggler, G. Gallego, and D. Scaramuzza. "Continuous-Time Trajectory Estimation for Event-based Vision Sensors". In: *Robotics: Science and Systems (RSS)*. 2015. DOI: 10.15607/RSS.2015.XI.036



(a) Continuous vs. Discrete

(b) Cummulative Cubic Splines

Figure 2.10: Continuous-Time Trajectories. (*a*): While the frames and inertial measurements arrive at a constant frequency, events are transmitted asynchronously and at much higher rates. We model the camera trajectory as continuous in time, which allows direct integration of all measurements using their precise timestamps. (*b*): Geometric interpretation of the cubic spline interpolation. The cumulative formulation uses one absolute control pose $\mathrm{T}_{w,i-1}$ and three incremental control poses $\Omega_i, \Omega_{i+1}, \Omega_{i+2}$ to compute the interpolated pose $\mathrm{T}_{w,s}$.

2.5 Event-based Robot Control

In this section, we describe how event cameras can be used in *closed-loop* for robot control.

2.5.1 Paper J: Slot-Car Racing

(P10) T. Delbruck, M. Pfeiffer, R. Juston, G. Orchard, E. Müggler, A. Linares-Barranco, and M. W. Tilden. "Human vs. Computer Slot Car Racing using an Event and Frame-Based DAVIS Vision Sensor". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. Lisbon, Portugal, May 2015, pp. 2409–2412. DOI: 10.1109/ISCAS.2015.7169170

This paper describes an open-source implementation for racing human vs. computer on a slot car track. A DAVIS is mounted in "eye-of-god" view. Its image frames are only used for setup and are subsequently turned off because they are not needed. The dynamic vision sensor (DVS) events are then used to track both the human and computer controlled cars. The precise control of throttle and braking afforded by the low latency of the sensor output enables consistent out-performance of human drivers at a laptop CPU load of <3 % and update rate of 666 Hz. The sparse output of the DVS event stream results in a data rate that is about 1000 times smaller than that from a frame-based camera with the same resolution and update rate. The scaled average lap speed of the 1/64 scale cars is about 450 km/h which is twice as fast as the fastest Formula 1 lap speed.

Related Software

(S3) https://github.com/tpietzsch/jAER/ ... /virtualslotcar

Related Videos

(V7) https://youtu.be/CnGPGiZuFRI

(V8) https://youtu.be/AsO1TWS8_VA

Figure 2.11: Slot-Car Racing

2.6 Unrelated Contributions

List of contributions that were performed during the Ph.D., but are not related to event-based vision. They can be divided in vision-based quadrotor navigation and heterogeneous robot collaboration.

2.6.1 Quadrotor Navigation

These works present methods for visual navigation of quadrotors. We show how standard cameras can be used for various robotics tasks, such as stabilization, navigation, autonomous 3D mapping, and passing through narrow gaps.

(U1) M. Faessler, E. Mueggler, K. Schwabe, and D. Scaramuzza. "A Monocular Pose Estimation System based on Infrared LEDs". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2014, pp. 907–913. DOI: 10.1109/ICRA.2014.6906962

(U2) M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza. "Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor MAV". in: *J. Field Robot.* 33.4 (2016), pp. 431–450. ISSN: 1556-4967. DOI: 10.1002/rob.21581

(U3) D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza. "Aggressive Quadrotor Flight through Narrow Gaps with Onboard Sensing and Computing". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017

2.6.2 Heterogeneous Robot Collaboration

These works deal with the collaboration of aerial and ground robots. As these robots have complementary capabilities (payload, perspective, maneuverability, etc.), using a heterogeneous team of robots can increase task performance. For example, in a search-and-rescue scenario, the aerial robot can map the environment and find a fast and feasible path for the ground robot. Due to its payload and manipulation capabilities, the ground robot can then deliver aid packages that are too heavy for aerial robots.

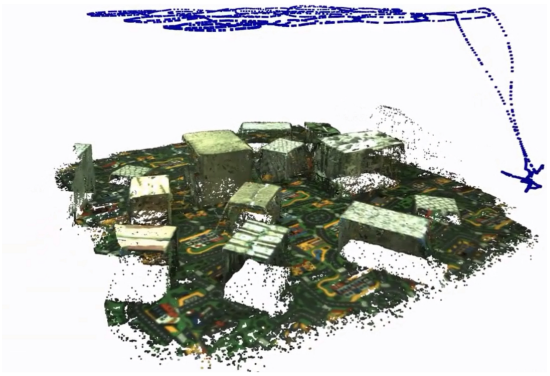
(U4) E. Mueggler, M. Faessler, F. Fontana, and D. Scaramuzza. "Aerial-guided Navigation of a Ground Robot among Movable Obstacles". In: *IEEE Int. Symp. Safety, Security, and Rescue Robot. (SSRR)*. 2014, pp. 1–8. DOI: 10.1109/SSRR.2014.7017662

(U5) R. Käslin, P. Fankhauser, E. Stumm, Z. Taylor, E. Mueggler, J. Delmerico, D. Scaramuzza, R. Siegwart, and M. Hutter. "Collaborative localization of aerial and ground robots through elevation maps". In: *IEEE Int. Symp. Safety, Security, and Rescue Robot. (SSRR)*. Oct. 2016, pp. 284–290. DOI: 10.1109/SSRR.2016.7784317

(U6) J. Delmerico, A. Giusti, E. Mueggler, L. M. Gambardella, and D. Scaramuzza. ""On-the-spot Training" for Terrain Classification in Autonomous Air-Ground Collaborative Teams". In: *Int. Symp. Experimental Robotics (ISER)*. 2016. DOI: 10.1007/978-3-319-50115-4_50

(U7) J. Delmerico, E. Mueggler, J. Nitsch, and D. Scaramuzza. "Active Autonomous Aerial Exploration for Ground Robot Path Planning". In: *IEEE Robot. Autom. Lett.* 2.2 (2017), pp. 664–671. DOI: 10.1109/LRA.2017.2651163

(a) U1: Monocular Pose Estimation using Infrared LEDs [46]

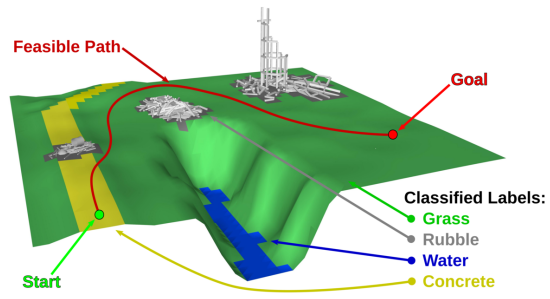


(b) U2: Live Dense 3D Reconstruction with an Autonomous Quadrotor [45]

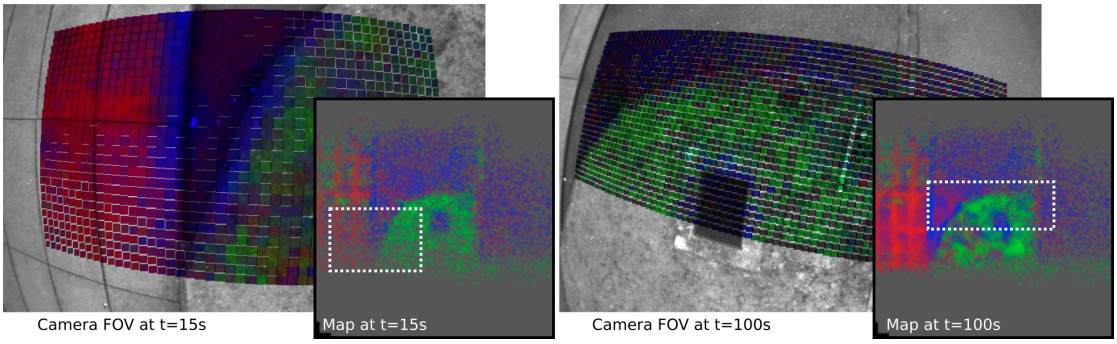
(c) U3: Quadrotor Flight through Narrow Gaps [47]

(d) U4: Aerial-guided Navigation of a Ground Robot among Movable Obstacles [98]

(e) U5: Collaborative Localization of Aerial and Ground Robots through Elevation Maps [64]



(f) U7: Active Autonomous Aerial Exploration for Ground Robot Path Planning [40]



(g) U6: "On-the-spot Training" for Terrain Classification [39]

Figure 2.12: Contributions to quadrotor navigation and heterogeneous robot collaboration.

3 Future Directions

While algorithms for event-based vision are still in an early stage, several works have demonstrated their advantages over high-quality standard cameras. The performance in terms of precision and robustness, however, must increase significantly to compete with existing approaches based on standard cameras. Partly, this requires increased sensor performance, where recent prototypes are very promising.

Increasing Sensor Performance. Current event cameras are still advanced prototypes and are not widely spread in the robotics and computer vision community. Several recent prototypes already increase the performance of event cameras in various regards. As is well known, one of the current limitations of event cameras is their limited spatial resolution (240×180 pixels, in the case of the DAVIS240C). Newer sensors, such as those presented in [76, 135] (still not commercially available), have a higher resolution: VGA (640×480 pixels). Such a resolution is suitable for robotics applications, but it is still small compared to the resolution of current standard cameras. Another limitation of event cameras, in the case of mobile robots such as lightweight quadrotors, is their size. Future devices, such as the mini-DAVIS from iniLabs, developed in the context of the DARPA Fast Lightweight Autonomy (FLA) program, aim at reducing the size of the sensor to be comparable to the size of standard cameras such as those found in smartphones.

Sensor performance is also expected to increase regarding brightness sensitivity and chromaticity (color is paramount in several computer vision and robotics applications such as segmentation and recognition). In [148], an event camera with higher sensitivity (1 % instead of typically 10–15 %) was presented. In [76], the C-DAVIS was introduced: a sensor that combines an event camera (QVGA resolution) with a standard color camera (VGA resolution). Hence, the C-DAVIS is able to capture spatial details with color and track movements with high temporal resolution while keeping the data output sparse and with low latency.

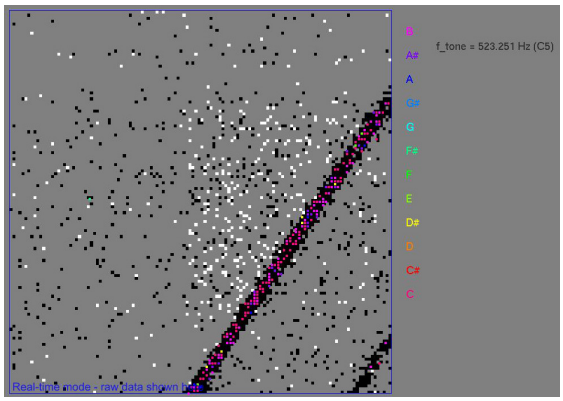


Figure 3.1: Detection of Piano String Frequency using an Event Camera

Integrated Processors. An additional advantage of event cameras is their low power consumption: the DVS requires 23 mW, whereas standard cameras typically require an order of magnitude more. However, current methods rely on standard processing hardware (either CPUs or GPUs) and, therefore, require much power for processing. Powerful processors, as the ones used for today's vision algorithms, use more than 100 W at full load. The human brain, for comparison, uses only 20 W [42]. Thus, a significant challenge will eventually be reducing power consumption of these algorithms. Neuromorphic processing hardware offers huge potential for low-power, massively parallel algorithms. Much power and resources are required to *transfer* data between two locations ("I/O bottleneck"). Thus, the earlier the data can be processed, the less energy is spent on it. An extreme form of this is presented in [44], where each pixel is equipped with an analogue processor. Thus, much computation can be performed with little power (in their case, they can perform 1.1 giga instructions per second at 40 mW).

Neuromorphic Processing Hardware. Dedicated hardware for neuromorphic processing is being developed, such as CAVIAR [131], SpiNNaker [51], IBM TrueNorth [3], and DYNAP [120]. Neuromorphic processing architectures allow more efficient handling of event-based data. While providing massive parallel compute power, they require very little power.

Neuromorphic Sensing. Besides event cameras, other sensor modalities are being researched using neuromorphic design principles. For example, in [84] a silicon cochlea is presented and event-driven touch sensors are shown in [25]. The iCub [94] is a neuromorphic humanoid robot equipped with event cameras and touch sensors.

Novel Applications for Vision. Due to its characteristics, event cameras open new applications for vision. For example, we developed a visual detector of string vibration

frequency (see Fig. 3.1).[1]

Summary

Event cameras offer great potential for many applications such as robotics and virtual reality because of their low latency, high dynamic range, and sparse output. Processing the visual input at the earliest possible stage allows reducing the required power. We believe that event cameras will become key to many applications and that we have just entered an avenue of endless research opportunities to design event-based algorithms and systems. The following appendices contain reprints of the papers that were discussed in Chapter 2.

[1] A video can be found on https://youtu.be/VgvN5vwXfq8

A Event-based Pose Tracking

E. Mueggler, B. Huber, and D. Scaramuzza. "Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2014, pp. 2761–2768. DOI: 10.1109/IROS.2014.6942940

Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers

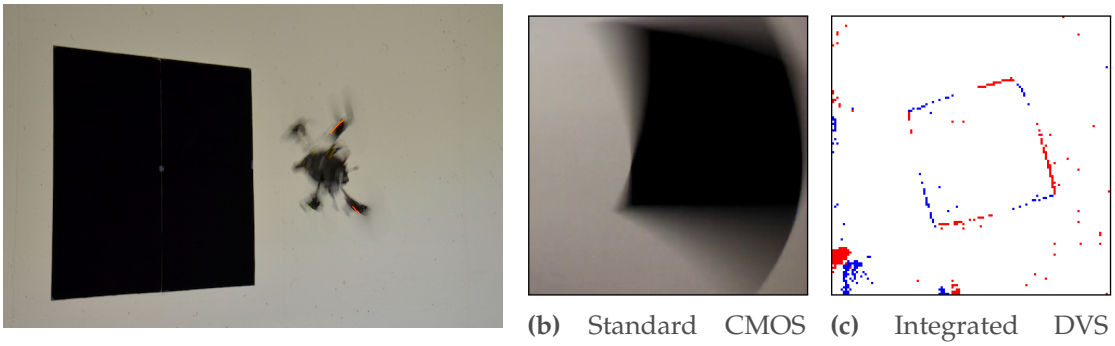
Elias Mueggler, Basil Huber and Davide Scaramuzza

Abstract — In the last few years, we have witnessed impressive demonstrations of aggressive flights and acrobatics using quadrotors. However, those robots are actually blind. They do not see by themselves, but through the "eyes" of an external motion capture system. Flight maneuvers using onboard sensors are still slow compared to those attainable with motion capture systems. At the current state, the agility of a robot is limited by the latency of its perception pipeline. To obtain more agile robots, we need to use faster sensors. In this paper, we present the first onboard perception system for 6-DOF localization during high-speed maneuvers using a Dynamic Vision Sensor (DVS). Unlike a standard CMOS camera, a DVS does not wastefully send full image frames at a fixed frame rate. Conversely, similar to the human eye, it only transmits pixel-level brightness changes at the time they occur with microsecond resolution, thus, offering the possibility to create a perception pipeline whose latency is negligible compared to the dynamics of the robot. We exploit these characteristics to estimate the pose of a quadrotor with respect to a known pattern during high-speed maneuvers, such as flips, with rotational speeds up to 1,200 $^\circ$/s. Additionally, we provide a versatile method to capture ground-truth data using a DVS.

A.1 Introduction

A.1.1 Motivation

In the last few years, impressive demonstrations of aggressive flight and acrobatics with quadrotors have been presented [93, 87]. Those systems are based on external motion-capture systems such as Vicon[1] or OptiTrack.[2] However, these setups are expensive, need active cameras, and are limited to small, confined workspaces. Thus, using onboard sensors is preferable for real-world applications. Many different sensor modalities have been proposed, such as laser scanners [132, 57], stereo cameras [133], and monocular cameras [147]. However, such systems achieve flight maneuvers that are still slow—especially in rotational speed—compared to those attainable with motion capture systems. Such high-speed performance is not achievable with commonly-used onboard sensors, such as CMOS cameras or laser range rangefinders.



(a) Our quadrotor performing a flip.

(b) Standard CMOS camera

(c) Integrated DVS events (2 ms)

Figure A.1: A quadrotor equipped with a standard CMOS camera and a DVS performing a flip. While the image of a standard CMOS camera suffers from high motion blur, a rendering of the DVS output shows that it can detect fast motion accurately. Blue and red indicate the polarity of the events (i.e., negative or positive changes of intensity).

The achievable agility of a robotic platform depends on the accuracy and latency of perception. The latency depends on the frequency of the sensor data, plus the time it takes to process the data. At the current state of the art, the latency of a CMOS-camera–based robot-perception pipeline is at the minimum in the order of 50–250 ms and the sampling rate in the order of 15–40 Hz. This puts a hard bound on the agility of the platform. We aim to overcome these limitations by exploiting a Dynamic Vision Sensor (DVS) [77]. Contrarily to standard frame-based CMOS cameras, which send entire images at fixed frame rates, a DVS only sends the local pixel-level changes caused by movement in a scene at the time they occur. The DVS output is a sequence of *asynchronous* events. Each pixel produces an event whenever it perceives a change of intensity. While the sensor's spatial resolution of 128×128 pixels is still

[1] http://www.vicon.com/
[2] http://www.naturalpoint.com/optitrack/

low, the temporal resolution is in the order of *microseconds*. Thus, we can achieve low-latency pose estimation even during very fast maneuvers, such as flips of a quadrotor (Figure A.1a). In addition, since a DVS only streams relative brightness *changes* in the sensor's field of view, the computational load can be reduced drastically. However, to take full advantage of the DVS capabilities, we must rethink the way we interpret visual data.

The method presented in this paper estimates the 6 Degrees-Of-Freedom (DOF) pose of a DVS with respect to a known passive pattern. A naïve solution would be to accumulate the events occurred over a certain time interval and adapt known pose-estimation algorithms for standard CMOS cameras to these "integrated" images (an example "integrated" image is shown in Figure A.1c, where we used an integration time of 2 ms). However, this is not desirable, because it would result in the same latency of a regular camera. Ideally, to have the lowest latency for the perception pipeline, one would want each single event to be reflected in small instantaneous changes of commands to the actuators. Therefore, we want to design methods that make use of the information contained in each single event. Since a DVS only detects changes of intensity, only scenes rich in gradient information are relevant. For simplicity, we chose a black square on a white background. However, our approach can be generalized to any planar shape or gradient map that is known a priori. Our algorithm starts by integrating events until the pattern is detected. Then, it tracks the line segments, which define the borders of the pattern, by updating both the lines and the pose at microsecond time resolution, as soon as a new event arrives.

A.1.2 Related Work

An impressive demonstration of the low-latency capabilities of a DVS for control applications was presented in [31]. Using two DVS, the authors implemented a pencil-balancing system on a highly-reactive platform free to move on a plane. The key to achieve such high-speed performance lies in an event-based adaptation of the Hough-transform line-detection algorithm [43] to track the pencil.

Asynchronous, event-based optical flow was presented in [10, 9]. The authors adapted the Lucas-Kanade tracking algorithm to cope with the event-based nature of the DVS.

An Event-based Iterative Closest Point Algorithm (ICP) was used in [108] for closed-loop control of a micro gripper. The mean update rate was 4 kHz. However, the algorithm integrates events over a predefined time interval and only works in 2D.

In our previous work [27], a DVS fixed to the ground was used to recover the pose of a quadrotor during flight by tracking LEDs mounted on the platform, which were blinking at very high frequencies. The DVS' time resolution allowed distinguishing different frequencies, thus avoiding the need for data association. While this system

successfully showed low-latency pose-tracking capabilities using a DVS, it required active markers (i.e., the blinking LEDs). Furthermore, the DVS was not mounted onboard the quadrotor.

Localization using a DVS on a ground robot was first presented in [145] and later extended to Simultaneous Localization And Mapping (SLAM) in [146]. However, the system was limited to planar motion and a 2D map. In their experiments, the authors used an upward-looking DVS mounted on a ground robot moving at low speed.

In our previous work [26], we presented a visual-odometry pipeline using a DVS in combination with a standard CMOS camera. We used a probabilistic framework that updates the pose likelihood relative to the previous CMOS frame by processing each event individually as soon as it arrives. As in [146], the experiments were performed at relatively low speeds (up to 30 °/s), while the system was limited to planar motion. Although higher speeds would in principle be possible, this was not feasible with those settings due to the occurrence of motion blur in the CMOS camera at higher speeds. In contrast, in this paper we focus on full 6-DOF pose estimation using only DVS input and demonstrate successful pose tracking at rotational speeds up to 1,200 °/s, such as during quadrotor flips.

A.1.3 Contributions and Outline

The main contribution of this paper is an event-based, low-latency method for 6-DOF localization that works for high-speed maneuvers, which we demonstrate during quadrotor flips. Additionally, we provide a versatile method to generate realistic datasets of simulated trajectories on artificial scenes with ground truth. Since we use the DVS in the loop, we can generate ground truth with real sensor noise.

The remainder of the paper is organized as follows. In Section A.2, we describe the DVS and a calibration procedure. Our algorithm is described in Section A.4 and evaluated in simulation and with real experiments in Section A.6.

A.2 Dynamic Vision Sensor

Standard CMOS cameras send full frames at fixed frame rates. On the other hand, retinal cameras such as a DVS have independent pixels that generate spike events at local relative brightness changes in continuous time. These events are timestamped and transmitted asynchronously at the time they occur using a sophisticated digital circuitry. Each event is a tuple $\langle x, y, t, p \rangle$, where x, y are the pixel coordinates of the event, t is the timestamp of the event, and $p \in \{-1, +1\}$ is the polarity of the event, which is the sign of the brightness change. This representation is sometimes also referred to as Address-Events Representation (AER). The DVS has a resolution of 128×128

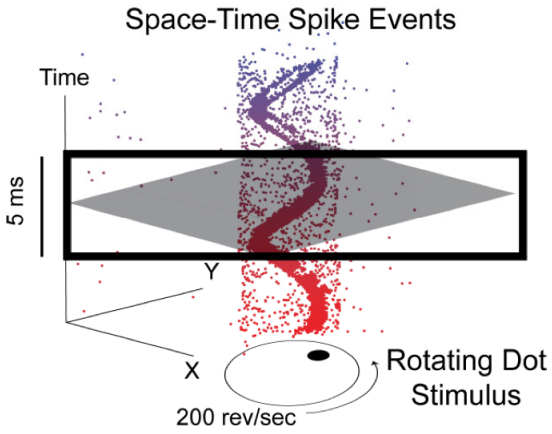


Figure A.2: Visualization of the output of a DVS looking at a rotating dot. Colored dots mark individual events. The polarity of the events is not shown. Events that are not part of the spiral are caused by sensor noise. Figure adapted from [82].

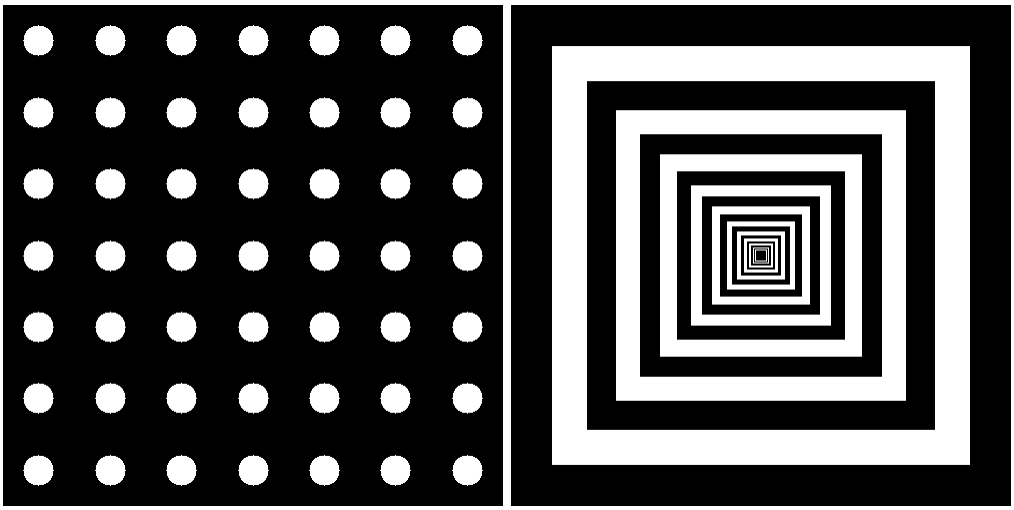
pixels and is connected via USB. A visualization of the output of the DVS is shown in Figure A.2.

A.3 Calibration

Since the optics of a DVS is the same as that of a regular camera, we use the standard pinhole camera model [136] to determine the intrinsic parameters (i.e., focal length, projection center, and distortion coefficients). For standard cameras, off-the-shelf calibration toolboxes based on regular patterns are the best choice [150]. However, it is not straightforward to use passive patterns with a DVS. Since relative motion is necessary to generate events, one would need to move the pattern in front of the DVS and integrate a sufficient number of events in order to "see" it.[3] Therefore, we calibrate the DVS using a computer screen with blinking patterns.[4] We use two different patterns: blinking dots (as depicted in Figures A.3a and A.8a) and concentric black-and-white squares (Figure A.3b). We use the former for intrinsic-parameter calibration (we utilize a standard calibration tool, such as [17]) and the latter for focus adjustment (we proceed by manually tuning the focus of the camera until the squares appear sharp). To be independent of the distance to the screen, we chose the squares to be spaced and scaled logarithmically.

[3]Remember that a DVS only generate asynchronous events; therefore, one would have to integrate the DVS events over a certain time interval in order to render an image that could be used with standard calibration tools.

[4]LED screens use pulse-width modulation of the background light for dimming. This high-frequency blinking generates events; thus, a static image on the screen appears blinking for the DVS.



(a) Intrinsic calibration **(b)** Focus calibration

Figure A.3: Calibration patterns for the DVS.

A.4 Event-based Pose Estimation

Since a DVS only detects changes of intensity, only scenes rich in gradient information are relevant. For simplicity, we chose a black square on a white background (Figure A.1). However, our approach can be generalized to any planar shape or gradient map that is known a priori. Our algorithm starts by integrating events until the pattern is detected. Then, it tracks the line segments, which define the borders of the pattern, by updating both the lines and the pose at microsecond time resolution, as soon as a new event arrives.

A.4.1 Initialization

Lines are detected using the Hough transform [43]. We chose the polar representation of lines and discretize the Hough space with equidistant bins of 7.5° and 2.5 pixels. Each event is added to the Hough space as it arrives. If a bin reaches a threshold of 25, it is considered a line candidate. If at least four distinct candidates are found, events are then assigned to each candidate based on their distance to the line. Events that are too far from the candidate line are removed. Then, all the events corresponding to a candidate line are ordered on the corresponding line (Figure A.4). If the gap between two consecutive events on the same line is too large (8 pixels), they are considered to belong to two different line segments. Only segments with a minimum length of 20 pixels are considered for the next step.

We perform an exhaustive search to find 4-sided shapes in the set of detected line segments. We start with one segment and append additional segments if they can be

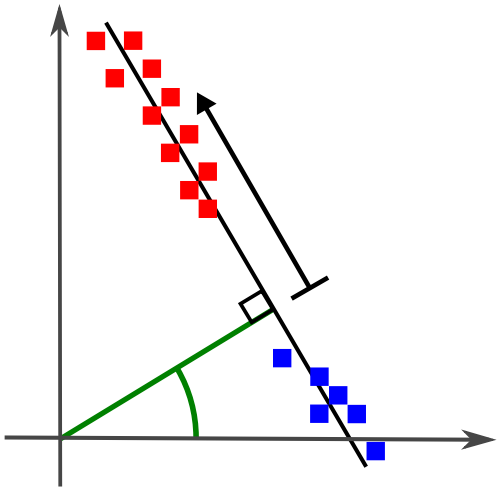


Figure A.4: Events (squares) belonging to a candidate line (black) detected in the Hough space (represented by r and θ). Events belonging to this line are ordered by their distance s and then clustered into line segments (e.g., red and blue). If a line segment is too short (less than 20 pixels), this is rejected by the algorithm (e.g., the blue cluster).

added in clockwise order and the angle is between 45° and 135°. If the fourth segment connects to the first one, the square is found. Then, we determine the four corners of the square by intersection of the estimated lines. Finally, we calculate the initial pose P from the homography relating the planar pattern and its image [59].

A.4.2 Line tracking

Lines are tracked in an event-based manner, which means, each event that arrives is used to directly update the pose estimate. When a new event arrives, we check whether it is close to one of the lines. If so, we use it to update that line and, subsequently, the pose estimate. Otherwise, we treat it as an outlier (i.e., the event was either generated by another object or by sensor noise) and reject it.

We represent each line with N past events. A new event replaces the *closest* one, as illustrated in Figure A.5. Note that always replacing the oldest event would eventually corrupt the line estimate as illustrated in Figure A.6. The choice of N is a tradeoff between latency and accuracy. While using many points would result in smoother trajectories, higher latency would be introduced. We found that $N = 8$ is good tradeoff for our setup.

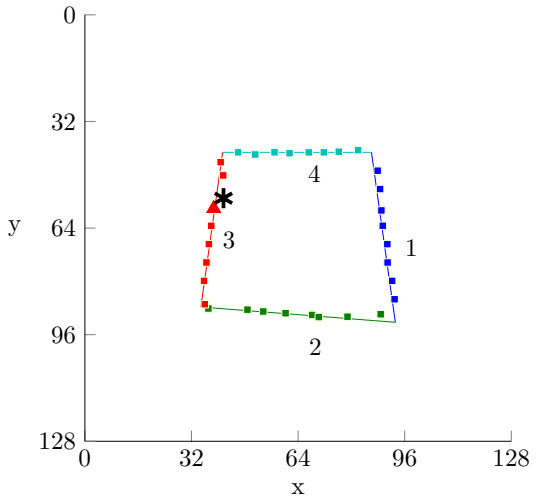
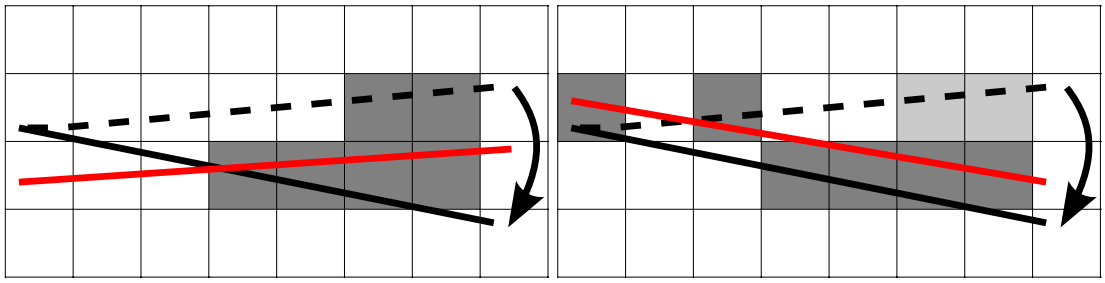


Figure A.5: Visualization of the tracking algorithm in the DVS image plane. A line is represented by 8 events (squares). When a new event (star) arrives, we check whether it is close to any line. If so, we replace the closest event with the new one. Otherwise, we treat it as an outlier and reject it. In this illustration, the event marked with the red triangle is replaced by the new event (represented by the star).



(a) Replacing oldest pixels **(b)** Replacing closest pixels

Figure A.6: Pixel-level schematics showing the problem of replacing the oldest event of a line during rotational motion. The true line (black dashed) is rotated (black solid). The line (red) is estimated by the events marked in gray. a Notice how replacing the oldest event shifts all events of a line towards one end, thus, corrupting the line estimate. b Instead, replacing the closest pixels does not suffer from this issue.

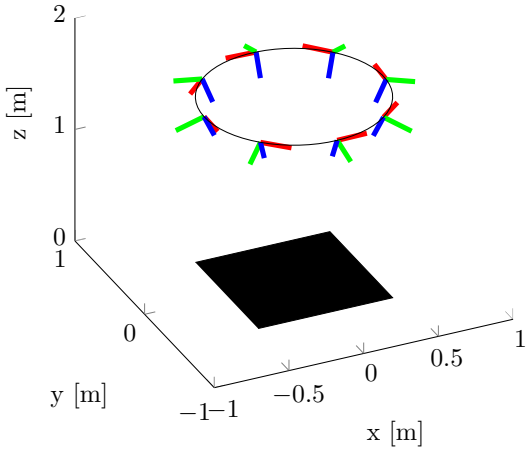


Figure A.7: Visualization of the pattern (black square) and the circular trajectory of the virtual camera.

A.4.3 Pose estimation

We update the pose by minimizing the sum of squared distances between the reprojection of each line and the events belonging to it, that is

$$P^* = \arg\min_P \sum_{l=1}^{4} \sum_{i=1}^{N} \| d\left(\pi(L_l, P), e_{l,i}\right) \|^2, \tag{A.1}$$

where L_l denotes a line belonging to the pattern, $\pi(\cdot,\cdot)$ projects a line onto the image plane, $e_{l,i}$ denotes an event i belonging to line l, and $d(\cdot,\cdot)$ returns the distance between the point and the line. The lines are updated with the new pose estimate P^* by projecting the pattern onto the image plane.

A.5 DVS Simulation

To assess the quality of our pose estimation algorithm, we need datasets with ground truth. To do this, we generated virtual camera views on a computer screen by simulating trajectories of a camera moving in front of a pattern, as depicted in Figure A.7. Instead of simulating a DVS output (which is not trivial given the sophisticated digital circuitry of a DVS), we placed a real DVS in front the screen and recorded the generated artificial views (Figure A.8). Having the DVS in the loop has the advantage that the sensor noise levels are real.

We denote the world frame of the artificial scene with a subscript W, the virtual camera frame with V, the computer screen frame with S, and the DVS frame with D. A world

(a) Calibration **(b)** Rendered scene

Figure A.8: Setup to simulate artificial scenes with ground truth. a A regular pattern is used to estimate the perspective transform between the DVS and the screen b The scene is rendered from a virtual camera that follows a given trajectory.

point X_W is mapped onto the virtual camera through a perspective transformation:

$$X_V = K_V \left(R_{VW} X_W + T_{VW} \right). \tag{A.2}$$

Since the output of the virtual camera is independent of the screen size, a scale factor α is introduced,

$$X_S = K_S X_V = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} X_V. \tag{A.3}$$

Because the screen and the DVS are not aligned, screen points are also mapped through a perspective transformation:

$$X_D = K_D \left(R_{DS} X_S + T_{DS} \right). \tag{A.4}$$

Substituting (A.2) and (A.3) into (A.4) gives

$$X_D = K_D \left(R_{DS} K_S K_V \left(R_{VW} X_W + T_{VW} \right) + T_{DS} \right), \tag{A.5}$$

where the virtual camera trajectory with respect to the world frame $P_{VW}(t) = [R_{VW}(t)|T_{VW}(t)]$ is a continuous function of time t.

We estimated the pose of the DVS with respect to the screen automatically before each recording, using a blinking pattern as described in Section A.3.

The scene was rendered in real-time using OpenGL.[5] For each event from the DVS, we

[5] http://www.opengl.org/

evaluated the virtual camera pose $[R_{VW}(t)|T_{VW}(t)]$ at the specific event time. Thus, we know the ground truth DVS pose for each event.

A.6 Experimental Evaluation

We evaluated our algorithm both with simulated data and real data from a quadrotor performing flips. In the evaluation, we used the angle of the angle-axis representation as an error metric for orientation.

A.6.1 Simulated Data

We used the simulation setup described in Section A.5. We simulated a planar scene containing a single black square on the $x-y$ plane centered in the origin of the world frame on a white background, as depicted in Figure A.7. We generated a circular trajectory at constant altitude z and commanded the angular velocity of the virtual camera such that its optical axis always intersected the origin of the world frame, that is:

$$P_{VW}(t) = \left[\begin{array}{ccc|c} c(\alpha) & s(\alpha) & 0 & 0 \\ -s(\alpha)c(\gamma) & c(\alpha)c(\gamma) & s(\gamma) & 0 \\ s(\alpha)s(\gamma) & -c(\alpha)s(\gamma) & c(\gamma) & z \end{array}\right],$$

where $s(\cdot) = \sin(\cdot)$, $c(\cdot) = \cos(\cdot)$, $\alpha(t) = 2\pi t/T$, $\gamma = 200^\circ$, $z = 1.7\,\text{m}$, and $T = 2\,\text{s}$ is the time it takes to complete a full circle. The square's side length is 0.9 m.

Figure A.9 shows the error of our pose estimation algorithm. The mean position error is 1.47 cm with a standard deviation of 0.72 cm. The mean orientation error is 2.28° with a standard deviation of 1.08°.

A.6.2 Real Data

Experimental Setup

We used a DVS with a 2.8 mm S-mount lens. We calibrated it as described in Section A.3 and found its focal length to be 69 pixels. We mounted the DVS on a Parrot AR.Drone 2.0 equipped with an Odroid U2 onboard computer (Figure A.10). The event stream was recorded onboard and streamed to a laptop over WiFi to visualize data in real-time. In addition to the DVS output, we also recorded the video of the front-looking standard CMOS camera.

As a pattern, we used a black square ($0.9 \times 0.9\,\text{m}$) attached to a white wall, the origin of the world frame coinciding with the center of the pattern, x being oriented

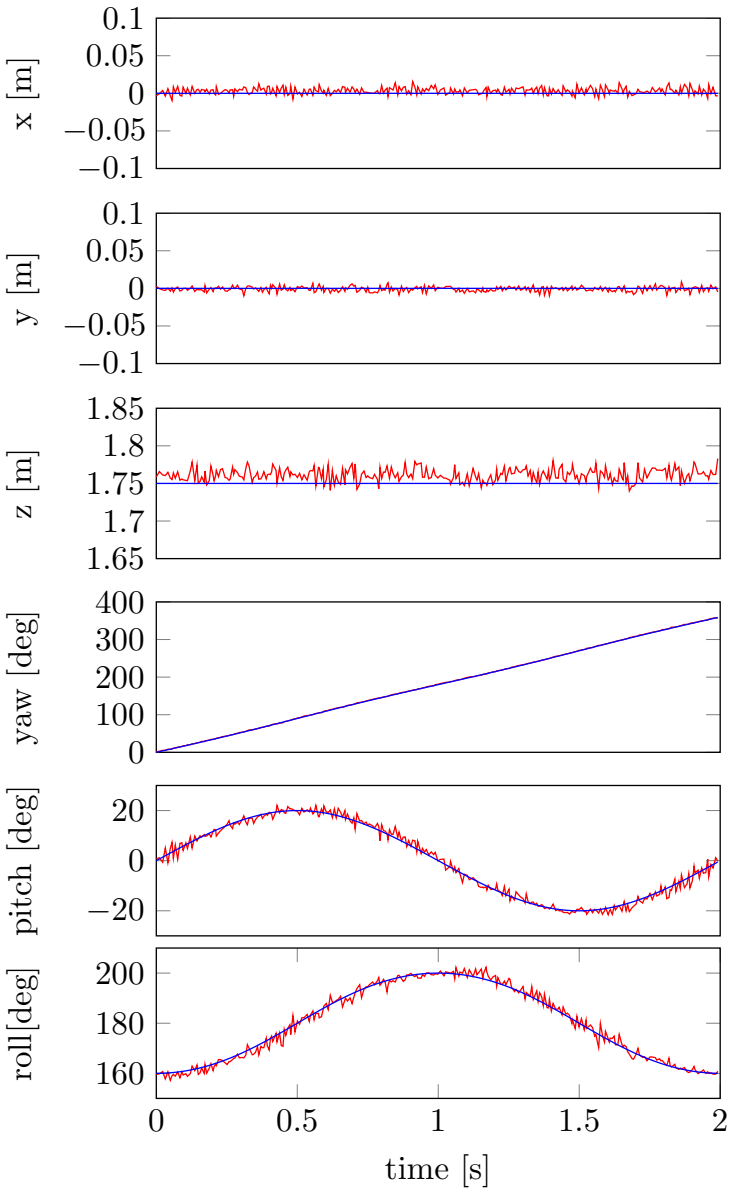


Figure A.9: Estimated trajectory (red) compared to ground truth (blue) on a simulated dataset. The trajectory is the one depicted in Figure A.7, which was generated as described in Section A.5.

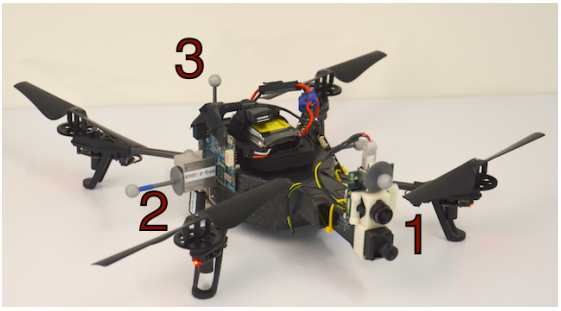


Figure A.10: Experimental setup on an AR.Drone. 1) The DVS (top) and a standard CMOS camera (bottom), 2) Odroid U2 computer for recording and streaming the DVS data over WiFi, and 3) markers to collect ground truth with a motion capture system.

perpendicularly to the wall and z parallel to the gravity vector.

Ground truth was captured using an OptiTrack motion capture system. Markers were placed all around the body of the quadrotor to ensure tracking during flips.

Evaluation

We controlled the quadrotor to perform multiple flips around the principal axis of the camera (roughly aligned with the x-axis of the world frame). The peak angular speed (i.e., roll rate) during such high-speed maneuvers was measured to be 1,200 °/s (cf. Figure A.15). While this results in severe motion blur effects for the standard CMOS camera (cf. Figure A.13), for the DVS we can still see very sharp lines if we integrate the events for an appropriate period of time (cf. Figure A.12). However, our algorithm does not rely on such integrated images, but updates the 6-DOF pose of the robot by processing each event individually as soon as it arrives. The estimated trajectory for three consecutive flips with ground truth is shown in Figure A.14. The mean position error is 10.8 cm with a standard deviation of 7.8 cm. The mean orientation error is 5.1° with a standard deviation of 2.4°.

During our experimental flight session, we recorded data for a total of 25 flips. Our algorithm could track the DVS trajectory for 24 of them (96 %). In only one case, tracking was lost during the flip. Figure A.11 shows the number of events as a function of the time during the first 15 flips. As observed, the density of events generated during flips is much larger than during near-hover flights.

Comparison with Theoretical Limit

Since our pose estimate is very noisy, we are interested to determine the accuracy of the pose estimate that one could achieve with an "ideal" CMOS camera (not a DVS)

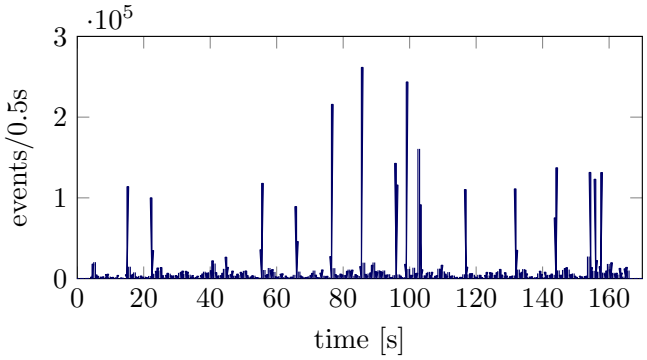
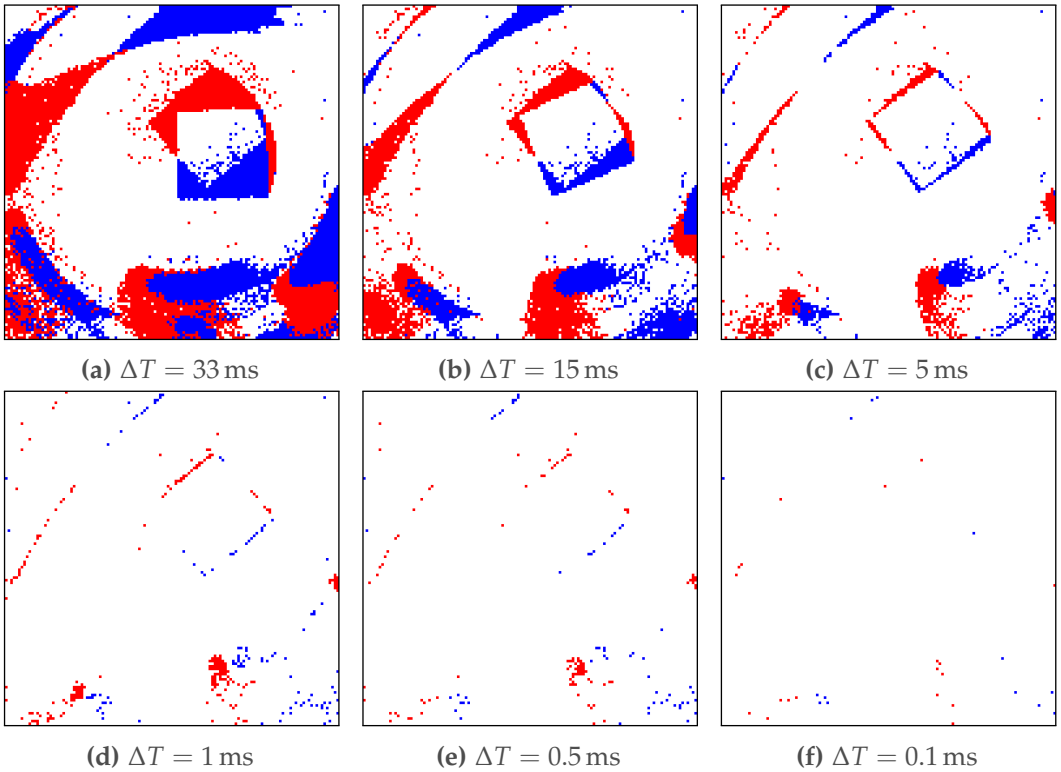


Figure A.11: Number of events as a function of time (we counted events in a time interval of 0.5 s) during an experimental session containing 15 flips. This plot clearly shows that during flips the density of events is much larger than during near-hover flights. During the first and last 5 s, the quadrotor is resting on the floor; thus, virtually no events are generated.



(a) $\Delta T = 33\,\text{ms}$ **(b)** $\Delta T = 15\,\text{ms}$ **(c)** $\Delta T = 5\,\text{ms}$

(d) $\Delta T = 1\,\text{ms}$ **(e)** $\Delta T = 0.5\,\text{ms}$ **(f)** $\Delta T = 0.1\,\text{ms}$

Figure A.12: Integrated events of the DVS over different time intervals. Blue and red indicate the polarity of the events.

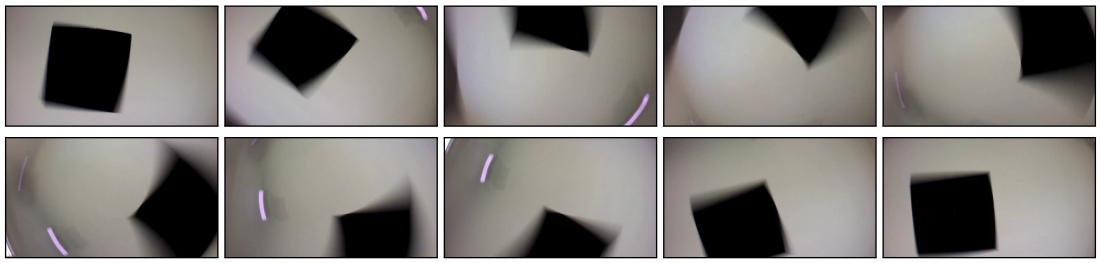


Figure A.13: Standard CMOS camera frames at 30 Hz during a flip (from left to right). Motion blur is clearly visible in all frames except the first and last one. The violet traces correspond to the LED lights of the OptiTrack cameras.

characterized by infinite frame rate and no motion blur, but having the same resolution as the DVS (i.e., 128×128 pixels). This problem is equivalent to characterize the pose estimation error of a CMOS camera in static settings in a configuration close to the real experimental setup (i.e., same intrinsic parameters, same pattern size, and same relative position between camera and pattern). Clearly, the answer depends on the accuracy (pixel or sub-pixel) of the edge detector. We addressed this by means of Monte-Carlo simulation, by adding Gaussian noise with different variances to all image points and by optimizing the pose by minimizing the reprojection error. We ran this simulation 1,000 times for each variance value.

The resulting error in position and orientation is shown in Figure A.16. The position and orientation accuracies of the DVS-based pose estimator described in this paper are indicated with horizontal red lines, corresponding to a mean position error of 10.8 cm and a mean orientation error of 5.1° respectively. As observed, these accuracies corresponds to a standard deviation of the error, which is in both cases smaller than 0.9 pixels. Since this can be considered as reasonably precise, we claim that the error of our DVS-based pose estimation is mainly caused by the poor resolution of the DVS (i.e., 128×128 pixels) and the results would significantly improve with a higher-resolution DVS.

A.7 Conclusion

In the last few years, we have witnessed impressive demonstrations of aggressive quadrotor flights and acrobatics using motion capture systems. Flight maneuvers using onboard sensors are still slow. At the current state, the agility of a robot is limited by the latency of its sensing pipeline. To obtain more agile robots, we need to use faster sensors. A Dynamic Vision Sensor (DVS) only transmits pixel-level brightness changes at the time they occur with microsecond resolution, thus, offering the possibility to create a perception pipeline whose latency is negligible compared to the dynamics of the robot. This technology is the most promising candidate for enabling highly

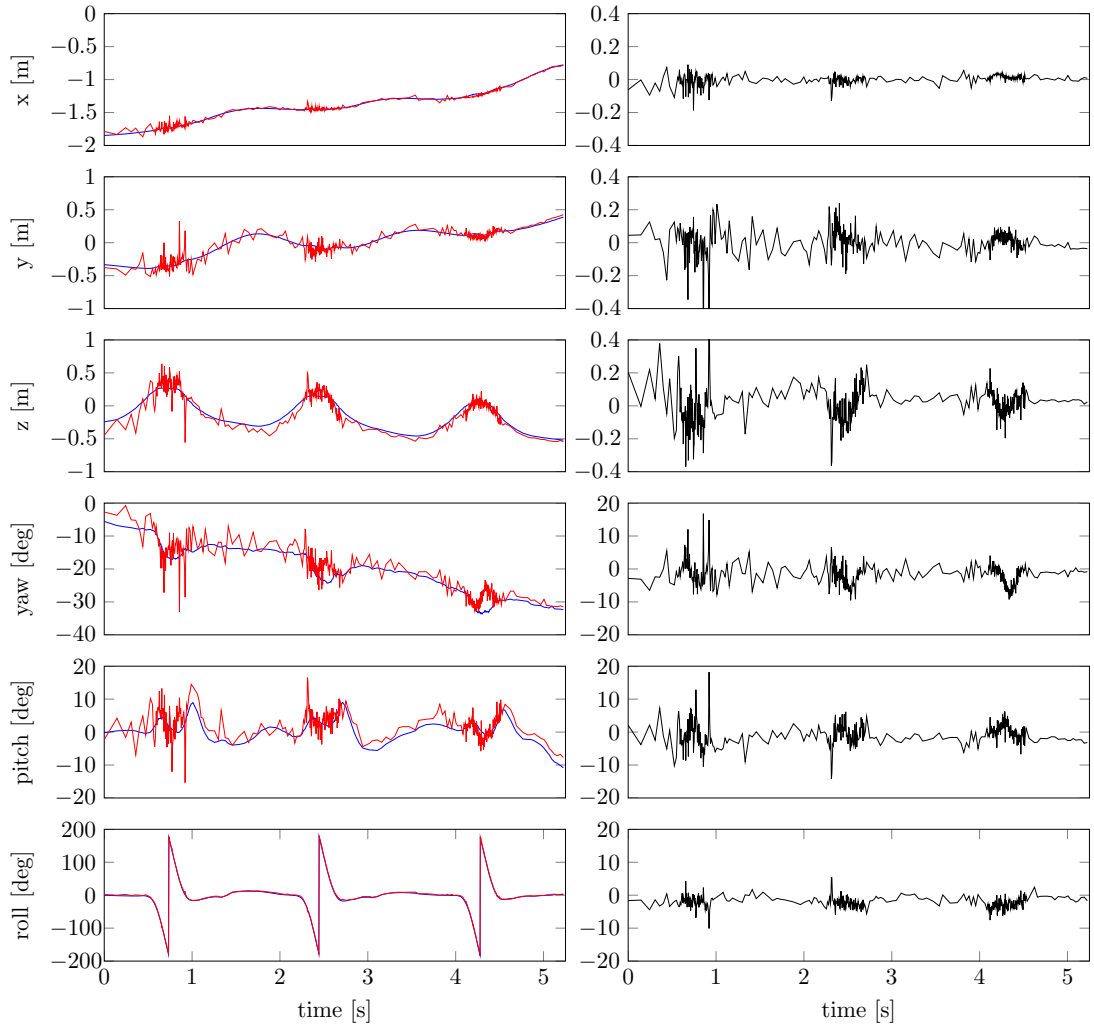


Figure A.14: Estimated trajectory (red) with ground truth (blue) and errors (black) for three consecutive flips with a quadrotor. Notice how the error gets smaller towards the end of the trajectory. This happens because the quadrotor moves closer to the pattern, which in turn appears larger in the DVS. Also notice how the density of pose estimates gets higher during flips. This occurs because the pose is updated whenever a new event arrives and the number of events increases during faster relative motion (cf. Figure A.11).

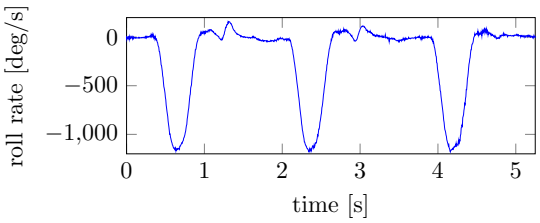
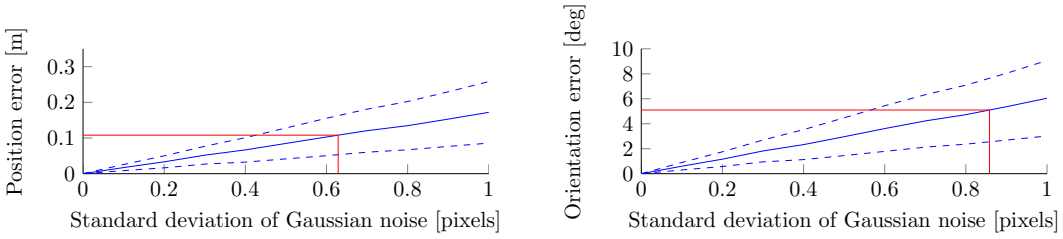


Figure A.15: Roll rate of the trajectory shown in Figure A.14. The maximum roll rate is 1,200 °/s during a flip.



(a) Position error **(b)** Orientation error

Figure A.16: Plots of mean pose error (solid) ± one standard deviation (dashed) in world coordinates for an ideal sensor with Gaussian noise in the image plane. The configuration is close to the experimental setup presented in Section A.6.2. The mean position and orientation error of the quadrotor flip experiment is marked in red and corresponds to 0.63 and 0.86 pixels, respectively.

aggressive autonomous maneuvers with flying robots. The current DVS prototypes suffer from a relatively poor resolution, which is currently being worked upon. In this paper, we presented the first onboard perception system for 6-DOF localization during high-speed maneuvers using a DVS. We demonstrated robust motion tracking during quadrotor flips with angular speeds up to 1,200 °/s. Future work will involve a generalization of the approach to arbitrary environments and the use of the DVS in closed-loop control.

Acknowledgement

We gratefully acknowledge the contribution of Flavio Fontana and Matthias Faessler for helping with the quadrotor experiments. We would also like to thank Tobi Delbruck and Vicente Villeneuva for helping us making the DVS lightweight enough for our experiments.

B Towards Evasive Maneuvers with Quadrotors

E. Mueggler, N. Baumli, F. Fontana, and D. Scaramuzza. "Towards Evasive Maneuvers with Quadrotors using Dynamic Vision Sensors". In: *Eur. Conf. Mobile Robots (ECMR)*. 2015, pp. 1–8. DOI: 10.1109/ECMR.2015.7324048

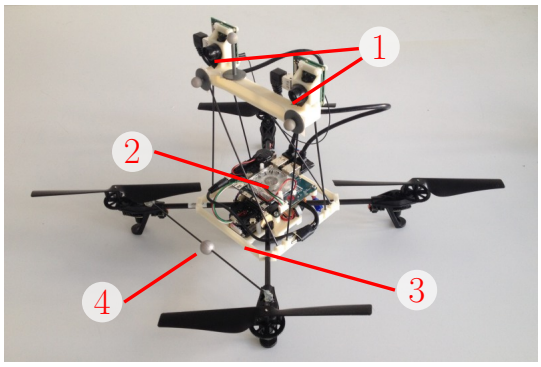
Towards Evasive Maneuvers with Quadrotors using Dynamic Vision Sensors

Elias Mueggler, Nathan Baumli, Flavio Fontana and Davide Scaramuzza

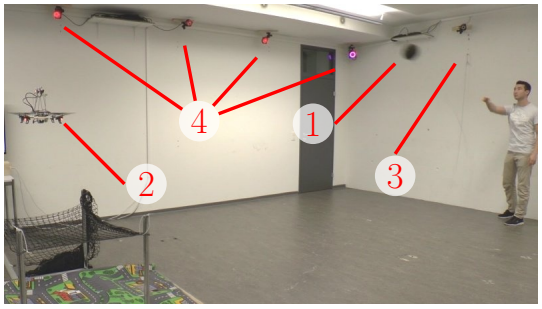
Abstract — We present a method to predict collisions with objects thrown at a quadrotor using a pair of dynamic vision sensors (DVS). Due to the *micro*-second temporal resolution of these sensors and the sparsity of their output, the object's trajectory can be estimated with minimal latency. Unlike standard cameras that send frames at a fixed frame rate, a DVS only transmits pixel-level brightness *changes* ("events") at the time they occur. Our method tracks spherical objects on the image plane using probabilistic trackers that are updated with each incoming event. The object's trajectory is estimated using an Extended Kalman Filter with a mixed state space that allows incorporation of both the object's dynamics and the measurement noise in the image plane. Using error-propagation techniques, we predict a collision if the 3σ-ellipsoid along the predicted trajectory intersects with a safety sphere around the quadrotor. We experimentally demonstrate that our method allows initiating evasive maneuvers early enough to avoid collisions.

B.1 Introduction

Collision avoidance of fast-moving objects requires high-frequency and low-latency sensors, algorithms, and control strategies. As an example, consider an object that is thrown at a robot at 30 m/s (i.e., 108 km/h) from a distance of 5 m, only 0.17 s are left to (i) detect the object, (ii) predict its trajectory, (iii) foresee if a collision will occur, and if so, (iv) initiate and (v) execute an evasive maneuver. Due to the inertia of the robot and its actuators, most of this time is required for the evasive action and only a small fraction (in the order of 10 ms) can be used for sensing and computation. During



(a) Quadrotor platform: (1) stereo DVS rig, (2) smartphone computer, (3) down-looking camera for vision-based stabilization, and (4) markers for ground truth with a motion-capture system. The details of our quadrotor platform are provided in [45].



(b) Experimental setup: (1) thrown ball, (2) quadrotor, (3) leash to avoid actual collisions, and (4) motion-capture system for ground-truth measurements.

Figure B.1: A ball is thrown towards an autonomous, vision-based quadrotor. The ball is detected and tracked using a pair of Dynamic Vision Sensors in a stereo configuration. Our algorithm predicts whether a collision will occur and can be used to initiate evasive maneuvers. The motion-capture system was only used to record ground-truth data of the ball and the quadrotor.

this time, a high-frequency camera running at 100 Hz would capture only one or two images, giving very limited data for the subtasks (ii) and (iii).

To achieve higher measurement frequencies while keeping the computational load small, new vision sensors are required. In this paper, we propose the use of Dynamic Vision Sensors (DVS) [77]. Contrarily to standard frame-based cameras that send entire images at fixed frame rates, a DVS only sends the local pixel-level brightness *changes* at the time they occur. These changes, which we call "events", are transmitted asynchronously and with low latency. While the sensor's spatial resolution of 128×128 pixels is still low, the temporal resolution is in the order of *micro*-seconds.

In the last few years, impressive demonstrations of aggressive flight and acrobatics with quadrotors have been presented [93, 86]. Among these demonstrations were also interactions with other, fast-moving objects, e.g. juggling balls [105], pole acrobatics [22], or flying through thrown circular hoops [92]. However, all these demonstrations are based on external motion-capture systems that track both the quadrotor and all other moving objects with very high precision and frequency (typically about 200 Hz). To bring these capabilities outside of laboratory environments, we cannot rely on external systems. Therefore, all sensing and computation must be performed *onboard* the vehicle. Due to their low weight and power consumption, vision-based approaches have been successfully demonstrated for autonomous, infrastructure-free flight with quadrotors [45]. However, due to motion blur at high speeds and computational

complexity for high frame rates, current vision-based quadrotor systems fly at relatively low speeds and only navigate in static environments.

In this paper, we present a method to predict collisions with objects thrown at a quadrotor using a pair of DVS in a stereo configuration (see Fig. B.1). To avoid a collision, a series of steps must be executed: (i) the thrown object must be detected, (ii) it must be tracked precisely to (iii) propagate its trajectory in time. Then, (iv) a decision on the action must be made to (v) initiate and execute an evasive maneuver. Possible applications are quick avoidance of other, uncooperative aerial vehicles and the escape of bird attacks[1].

The remainder of the paper is organized as follows. In Section B.2, we review related work. The DVS is described in Section B.3, followed by an evaluation of the latencies of both standard frame-based cameras and the DVS in Section B.4. Our algorithm is described in Section B.5 and experimentally evaluated in Section B.6.

B.2 Related Work

An impressive demonstration of the low-latency capabilities of a DVS for control applications was presented in [31]. Using two DVS, the authors implemented a pencil-balancing system on a highly-reactive platform free to move on a plane. A robotic goalkeeper with a reaction time of 3 ms was presented in [34].

An Event-based Iterative Closest Point Algorithm (ICP) was used in [108] for closed-loop control of a micro gripper. The mean update rate was 4 kHz. The algorithm integrates events over a predefined time interval and only works in 2D.

Asynchronous, event-based optical flow was presented in [9]. The authors adapted the Lucas-Kanade tracking algorithm to cope with the event-based nature of the DVS. The event-based optical flow was later used [28] for event-based computation of the time-to-contact [73]. This approach, however, assumes that the trajectories of the robot and the obstacles are aligned, i.e., when the robot continues to move, a collision is unavoidable. In this paper, we explicitly estimate the trajectory of the thrown object, since it might no intersect with the robot and no evasive action is required.

Several approaches of event-based stereo matching can be found in the literature. In [11], the high temporal resolution of the DVS was exploited for stereo matching. In [74], event histograms were used for stereo correspondence. The output of the algorithm was used for gesture recognition. In [24], six synchronized DVS were used for 3D reconstruction using N-ocular stereo vision.

[1] See, e.g., http://youtu.be/DzfiLmbhvqg or http://youtu.be/smv7cBzg-Ok.

In our previous work [27], a DVS fixed to the ground was used to recover the pose of a quadrotor during flight by tracking LEDs mounted on the platform, which were blinking at very high frequencies. The DVS' time resolution allowed distinguishing different frequencies, thus avoiding the need for data association. While this system successfully showed low-latency pose-tracking capabilities using a DVS, it required active markers (i.e., the blinking LEDs). Furthermore, the DVS was not mounted onboard the quadrotor. We use a similar concept for intrinsic and extrinsic camera calibration.

Localization using a DVS on a ground robot was first presented in [145] and later extended to Simultaneous Localization And Mapping (SLAM) in [146]. However, the system was limited to planar motion and a 2D map. In their experiments, the authors used an upward-looking DVS mounted on a ground robot moving at low speed.

In previous work, we showed how a DVS can be used onboard a flying robot for localization during high-speed maneuvers [102], where rotational speeds of up to 1,200 °/s were measured during quadrotor flips.

B.3 Dynamic Vision Sensors

B.3.1 Working Principle

Standard CMOS cameras send full frames at fixed frame rates. On the other hand, event-based (retinal) cameras such as the DVS [77] have independent pixels that generate events at local relative brightness changes in continuous time. These events are timestamped and transmitted asynchronously at the time they occur using sophisticated digital circuitry. Each event e is a tuple $\langle \mathbf{p}, t, p \rangle$, where $\mathbf{p} = (x, y)$ are the pixel coordinates of the event, t is the timestamp of the event, and $p \in \{-1, +1\}$ is the polarity of the event, which is the sign of the brightness change. This representation is sometimes also referred to as Address-Events Representation (AER). The DVS has a resolution of 128×128 pixels and is connected via USB. A visualization of the output of the DVS is shown in Fig. B.2.

Due to its low latency and high temporal resolution, both in the range of *micro*-seconds, the DVS is a very promising sensor for high-speed mobile robot applications. Since the data stream from the DVS is sparse (only *changes* are reported), the bandwidth and computational load are low. An additional advantage for robotic applications is the DVS' high dynamic range of 120 dB (compared to 60 dB of expensive computer-vision cameras), which allows both indoor and outdoor operation without changing parameters. Since all pixels are independent, these contrasts can also take place within the same scene.

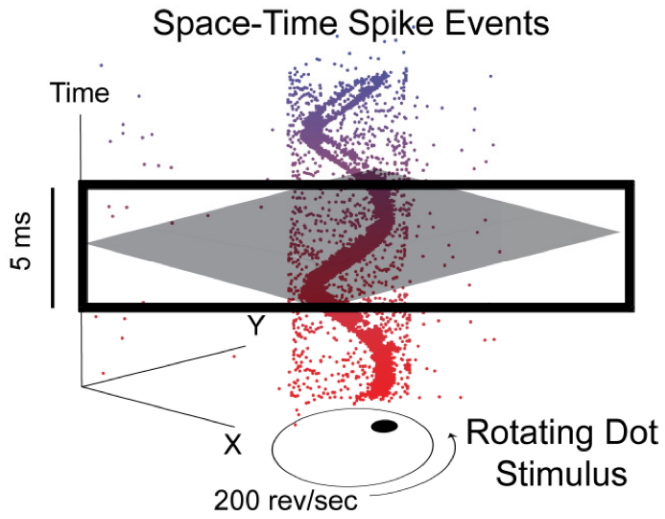


Figure B.2: Visualization of the output of a DVS looking at a rotating dot. Colored dots mark individual events. The polarity of the events is not shown. Events that are not part of the spiral are caused by sensor noise. Figure adapted from [82].

B.3.2 Calibration

We used a board with blinking LEDs for intrinsic and extrinsic calibration of the DVS stereo setup (see Fig. B.3). While we used computer screens in previous work [102] for calibration, we found that an LED checkerboard allows for larger viewing angles and, thus, better calibration results. Since a DVS only responds to changes in the scene, blinking LED allow us to artificially trigger events without moving the sensor. Due to its very high temporal resolution, a DVS can easily cope with the LED blinking frequency of 1 kHz. As shown in [27], this frequency is well above those generated by moving the sensor or moving objects in the scene. Since the optics are the same for the DVS as for frame-based cameras, we can rely on standard algorithms for intrinsic and extrinsic calibration. We released our ROS-compatible[2] DVS driver and calibration suite as open-source software[3].

B.4 Sensor Latencies

To motivate the use of DVS for low-latency and high-speed robotic applications, we compare its latency to frame-based cameras. To do so, we measure the round-trip delay between toggling an LED and the detection of this change by the different sensors (see Fig. B.4). This measurement includes sending the command to toggle the LED, the time to capture an image (in the frame-based case), data transfer to the computer, and simple computations to detect the change.

[2]Robot Operating System, http://www.ros.org
[3]http://www.github.com/uzh-rpg/rpg_dvs_ros

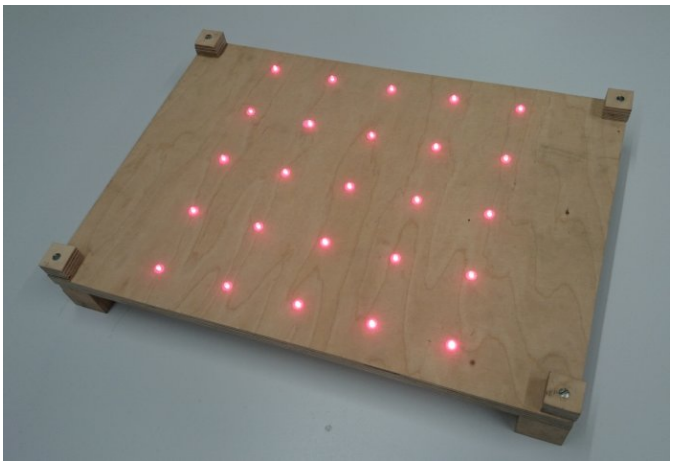


Figure B.3: Board with blinking LEDs for intrinsic and extrinsic calibration of the DVS stereo setup. The LEDs are blinking at a frequencies of 1 kHz, such that they can easily be detected by a DVS.

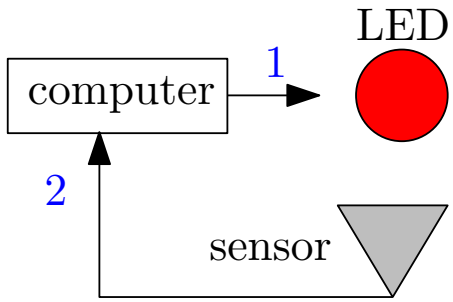


Figure B.4: Picture of sensor-latency measurement setup: the LED (1) is triggered by a computer and observed by a sensor (2). In our case, the sensor is either a DVS, a BlueFOX, or an ASUS Xtion. We measure the round-trip delay from sending the signal until the change was detected by the sensor. The experiments were performed for each sensor individually.

B.4.1 Experimental Setup

We compare the DVS with two frame-based cameras: the ASUS Xtion Pro Live and MatrixVision mvBlueFOX-MLC200w. The ASUS Xtion has a rolling shutter, a resolution of 640×480 pixels, and provides RGB-D images at 30 Hz. Exposure and gain are controlled by the sensor automatically. The BlueFOX camera has a global-shutter, a resolution of 752×480 pixels, and provides grayscale images up to 90 Hz. The exposure time was set to 4 ms and the gain to 0 dB. The bias-generation setting for the DVS were set to "fast"[4]. We interfaced all sensors over USB and evaluated the delays both on a laptop computer (Lenovo W530) and an embedded computer (Hardkernel Odroid U3).

Back-of-the-envelope calculations for the two frame-based cameras show the minimally

[4] http://sourceforge.net/p/jaer/code/HEAD/tree/jAER/trunk/biasgenSettings/DVS128/DVS128Fast.xml

achievable latencies for this setup: USB 2.0 is specified for 480 Mbit/s. An RGB image from the ASUS Xtion has a raw size of $3 \times 640 \times 480 \times 8$ bits $= 7.3$ Mbit, yielding a transfer duration of 15.2 ms. A grayscale image from the BlueFOX has a raw size of $752 \times 480 \times 8$ bits $= 2.1$ Mbit, yielding a transfer duration of 6.0 ms. This is only a lower bound on the transfer duration: the exposure time must also be added. An event of a DVS is encoded in 32 bit, yielding a transfer duration of 0.067 µs. Therefore, the event rate is bounded by 15 million events per second.

An LED is triggered using a PX4FMU-Autopilot board[5], which is the same board that interfaces the motor controllers on many quadrotor platforms. It is optimized for reliable and low-latency communication.

To detect the LED on the frame-based cameras, we defined a small region of interest in which the LED is visible. We then computed the mean intensity in that region and reported a detection when the mean changed by more than a threshold.

To detect the LED using the DVS, we measured the number of events per time unit. When nothing changes, only "background-activity" events are transmitted, which are caused by sensor imperfections. However, as soon as the LED is toggled, many events are generated.

For each combination of computer, toggle direction, and sensor, we collected more than 1,000 measurements. To avoid aliasing effects, we waited for a random amount of time after each detection before toggling the LED again. No triggering signal was missed in all the experiments.

B.4.2 Results

The sensor latency (i.e., capturing the image, transferring it to the host computer, and performing a simple computation to detect a change) is assumed to be Gaussian. It is modeled with a random variable $X_1 = \mathcal{N}(\mu, \sigma)$, where μ is the mean delay and σ is the standard deviation. Due to the fixed frequency of standard cameras (typically $f = 30$ Hz), a uniformly-distributed delay between 0 and $\Delta t = 1/f$ is added to the sensor latency: $X_2 = \mathcal{U}(0, \Delta t)$, where Δt is the time between two frames. Therefore, the expected delay Y is thus a sum of two independent random variables, $Y = X_1 + X_2$. We identify the mean μ and standard deviation σ of X_1 in terms of those of Y and X_2, and replacing the mean and variance of Y by their empirical values (sample mean and sample variance), as detailed in the Appendix. For the DVS, only a Gaussian was fitted because it works asynchronously.

A histogram comparing the latency distributions of the DVS and the BlueFOX camera is shown in Fig. B.5. While the delays of the DVS are an order of magnitude lower, it

[5] http://www.pixhawk.org/modules/px4fmu

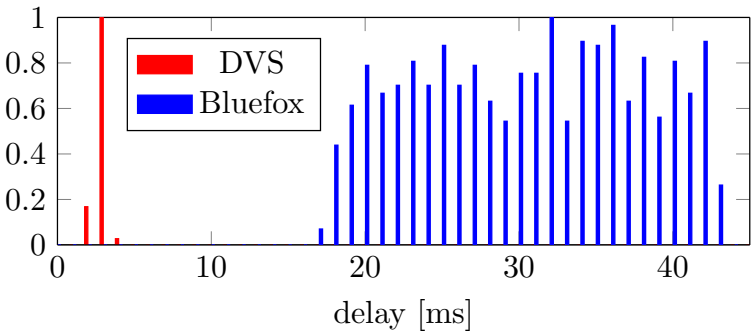


Figure B.5: Round-trip delays for a DVS (red) and a BlueFOX camera (blue) using a laptop computer. The delays for the BlueFOX camera are wide spread due to the synchronous nature of the sensor. In our experiments, it ran at 40 Hz, which corresponds to 25 ms. The histograms were normalized with their maximum value.

Table B.1: Round-trip delays using a laptop computer

Sensor	μ_{on}	σ_{on}	μ_{off}	σ_{off}	Δt	Unit
ASUS Xtion	28.0	2.7	50.3	2.8	33	ms
BlueFOX	14.5	0.5	18.0	0.5	25	ms
DVS	2.8	0.3	2.8	0.3	0	ms

Table B.2: Round-trip delays using an embedded computer

Sensor	μ_{on}	σ_{on}	μ_{off}	σ_{off}	Δt	Unit
ASUS Xtion	45.0	4.0	67.3	2.8	33	ms
BlueFOX	19.9	0.4	20.5	1.2	25	ms
DVS	4.5	1.2	4.2	0.8	0	ms

can also be seen that the delays of the BlueFOX camera are wide spread due to the synchronous nature of the sensor. In our experiments, it was running at a framerate of 40 Hz, which corresponds to 25 ms.

In Table B.1, we summarize the round-trip delays for the three vision sensors when connected to the laptop computer. We report the identified parameters of the histogram distribution, where the indices *on* and *off* indicate measurements when turning on and off the LED, respectively. Table B.2 provide the same results measured on the embedded computer. Our results are in line with the results reported in [34] of 2.2(20) ms.

B.5 Algorithm

In this section, we describe an algorithm to track spherical objects thrown at flying quadrotor using two DVS in a stereo configuration (see Fig. B.1a). We first describe an event-based circle tracker that is similar to the trackers introduced in [72]. Then, we match trackers from the left and right sensor that fulfill a set of geometric constraints. We exploit the high temporal resolution of the DVS to measure the disparity with sub-pixel accuracy using the correlation of spatio-temporal neighborhoods of matching trackers. These measurements are used to initialize and update an Extended Kalman Filter (EKF). A mixed state space allows incorporation of both the object's dynamics and the measurement noise on the image plane. We then propagate the current state of the system with its uncertainty in time and check for a collision of the 3σ-ellipsoid around the predicted trajectory with a safety sphere around the quadrotor. In the following, we detail these steps.

B.5.1 Event-based Circle Tracker

Our trackers are similar to the ones described in [72], but specialized to track spherical objects. We describe circular trackers by their mean position $\boldsymbol{\mu}_p$, radius $\boldsymbol{\mu}_r$, and uncertainty in radius σ_r as illustrated in Fig. B.6. For each incoming event, we evaluate its score $p_i(\mathbf{p})$ that belongs to tracker i,

$$p_i(\mathbf{p}) = \frac{1}{\sqrt{2\pi}\sigma_{r,i}} \exp\left(-\frac{1}{2}\left(\frac{d_i - \mu_{r,i}}{\sigma_{r,i}}\right)^2\right), \tag{B.1}$$

where $d_i = \|\mathbf{p} - \boldsymbol{\mu}_{p,i}\|$ is the distance between the event's position $\mathbf{p}$ and the tracker's position $\boldsymbol{\mu}_{p,i}$. The tracker $i = i_{max}$ with the highest score is then updated using an Infinite Impulse Response (IIR) filter,

$$\begin{aligned}
\boldsymbol{\mu}_p(t) &= \boldsymbol{\mu}_p(t_{\text{prev}}) + \alpha_p\left(\mathbf{p} - \boldsymbol{\mu}_p(t_{\text{prev}})\right), \\
\mu_r(t) &= \mu_r(t_{\text{prev}}) + \alpha_r\left(d - \mu_r(t_{\text{prev}})\right), \\
\sigma_r^2(t) &= \sigma_r^2(t_{\text{prev}}) + \alpha_\sigma\left((d - \mu_r(t))^2 - \sigma_r^2(t_{\text{prev}})\right),
\end{aligned} \tag{B.2}$$

which corresponds to an exponentially-weighted moving average filter with *smoothing factors* $\{\alpha_\text{p}, \alpha_r, \alpha_\sigma\} \in (0, 1)$. Small values indicate more smoothing but also induce more latency. In our experiments, we empirically set $\alpha_\text{p} = 0.01$, $\alpha_r = 0.002$, and $\alpha_\sigma = 0.005$.

The activity $\mathcal{A}_i$ of a tracker indicates when the tracker was last updated by events,

$$\mathcal{A}_i(t) = \begin{cases} \mathcal{A}_i(t - \Delta t)\exp\left(-\frac{\Delta t}{\tau}\right) + p_i(\mathbf{p}), & i = i_{max}, \\ \mathcal{A}_i(t - \Delta t)\exp\left(-\frac{\Delta t}{\tau}\right), & \text{otherwise.} \end{cases} \tag{B.3}$$

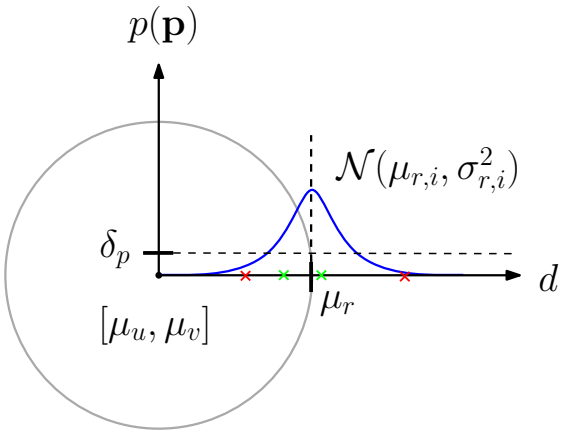


Figure B.6: Illustration of the Gaussian radius distribution $\mathcal{N}(\mu_r(t), \sigma_r^2(t))$ of a tracker. An event with a score p_i below δ_p, e.g., red crosses, is not considered, while an event above δ_p, e.g., green crosses, is considered to be generated by the circular shape.

In addition, each tracker has k directional activities a_j with $j = 1, \ldots, k$, which is defined as (B.3) but only for a $1/k$-th section of the circle. From these directional activities, we compute a factor γ,

$$\gamma = \frac{1}{\mathcal{A}_i} \sum_{j=1}^{k} \frac{a_j}{k}, \tag{B.4}$$

which is high if the events are distributed uniformly around the circle and low otherwise. Therefore, γ indicates how well the tracker follows an actual circle. In the following, we only consider trackers that have $\mathcal{A} > 15$ and $\gamma > 0.7$, i.e., trackers that have received sufficient support from the event stream and follow a circle. To avoid that the trackers follow arbitrary shapes, we restrict the radius, its variance, and the variance-to-mean ratio to reasonable intervals. For the initialization of the trackers and more details of the algorithm, we refer the reader to [72].

B.5.2 Stereo Matching

We match active trackers from the left and right sensor plane that fulfill all of these four constraints: (i) the disparity d must be positive, (ii) the mean vertical positions μ_x of the trackers must be close, (iii) the radius r of the trackers must be similar, and (iv) each tracker can only have one matching tracker.

Due to the low spatial resolution of the DVS of only 128×128 pixels and the short baseline of 12 cm, the disparity at 5 m is only 3.1 pixels. Thus, noise in the tracker positions has a significant impact on the accuracy of the depth measurement. We therefore refine the disparity estimate with a sub-pixel estimation algorithm that we

initialize with the stereo-matching estimate.

B.5.3 Sub-Pixel Disparity Estimation

To increase the precision of the stereo matching, we compute the correlation of the spatio-temporal neighborhood of the two matching trackers. More precisely, we correlate the Surface of Active Events (SAE) [10] using linear interpolation. The SAE $\Sigma_p(\mathbf{p})$ stores the last timestamp of an event with polarity p that was reported at pixel location $\mathbf{p}$,

$$\Sigma_p(\mathbf{p}) \leftarrow t. \tag{B.5}$$

We search for the maximum correlation of the left and right SAE,

$$d = \arg\max_d \sum_p \sum_u \sum_v \tilde{\Sigma}_p^l(u, v) \cdot \tilde{\Sigma}_p^r(u + d, v), \tag{B.6}$$

where l and r refer to the left and right SAE, respectively, and $\tilde{\Sigma}$ is the shifted SAE defined as

$$\tilde{\Sigma}(\mathbf{p}) = \max(\Sigma(\mathbf{p}) - t_{\text{curr}} + \Delta T, 0), \tag{B.7}$$

where t_{curr} is the current time and $\Delta T = 50\,\text{ms}$. All timestamps on the shifted SAE are between 0 and ΔT. We first evaluate (B.6) in 1-pixel steps and then refine it with 0.1-pixel steps. This results in a measurement of the ball's center of mass in the image plane, i.e., its location u, v and its disparity d with sub-pixel accuracy.

B.5.4 Extended Kalman Filter

Under the assumption of negligible air drag and a perfectly-stable hovering quadrotor (i.e., pitch, roll, and all angular rates are zero), the ballistic trajectory of a ball in the sensor's 3D coordinate frame is given by

$$\begin{bmatrix} X(t) \\ Y(t) \\ Z(t) \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} + \begin{bmatrix} v_{X0} \\ v_{Y0} \\ v_{Z0} \end{bmatrix} t + \frac{1}{2} \begin{bmatrix} 0 \\ g \\ 0 \end{bmatrix} t^2, \tag{B.8}$$

with initial position $[X_0, Y_0, Z_0]^\top$, initial velocity $[v_{X0}, v_{Y0}, v_{Z0}]^\top$, and gravitational acceleration g.

For the Extended Kalman Filter (EKF), we use a mixed state $\mathbf{x}(t)$ consisting of the ball's

position in image coordinates and its velocity in world coordinates,

$$\begin{aligned}\mathbf{x}(t) &= [x_1, x_2, x_3, x_4, x_5, x_6]^\top \\ &= [d, \dot{Z}, x_{\text{com}}, \dot{X}, y_{\text{com}}, \dot{Y}]^\top.\end{aligned} \tag{B.9}$$

This allows us to incorporate the measurement noise in pixel units with the dynamics of the ball (B.8). The coordinates in image space and world coordinates are linked by the pinhole camera model, i.e.,

$$X = b \cdot \frac{u}{d}, \quad Y = b \cdot \frac{v}{d}, \quad Z = b \cdot \frac{f}{d}, \tag{B.10}$$

where f is the focal length of the camera and b the baseline of the stereo setup.

Using (B.10) and (B.8), the nonlinear continuous-time system can be derived as

$$\dot{\mathbf{x}}(t) = \mathbf{q}\left(\mathbf{x}(t)\right) = \begin{bmatrix} \frac{-1}{fb} \cdot x_1^2 \cdot x_2 \\ 0 \\ \left(\frac{1}{b} \cdot x_4 - \frac{1}{fb} \cdot x_2 \cdot x_3\right) \cdot x_1 \\ 0 \\ \left(\frac{1}{b} x_6 - \frac{1}{fb} \cdot x_2 \cdot x_5\right) \cdot x_1 \\ g \end{bmatrix} \tag{B.11}$$

with observation vector

$$\mathbf{z}(t) = \mathbf{h}\left(\mathbf{x}(t)\right) = [x_1, x_3, x_5]^\top. \tag{B.12}$$

Using a first-order approximation of the derivate and a timestep of T_k, the discrete-time nonlinear system becomes

$$\mathbf{x}(t_{k+1}) = \mathbf{q}\left(\mathbf{x}(t_k)\right) = \begin{bmatrix} \left(\frac{-1}{fb} \cdot x_1^2 \cdot x_2\right) \cdot T_k + x_1 \\ x_2 \\ \left(\frac{x_4}{b} - \frac{1}{fb} x_2 \cdot x_3\right) \cdot x_1 \cdot T_k + x_3 \\ x_4 \\ \left(\frac{1}{b} x_6 - \frac{1}{fb} \cdot x_2 \cdot x_5\right) \cdot x_1 \cdot T_k + x_5 \\ g \cdot T_k + x_6 \end{bmatrix}.$$

The linearized system matrix $\mathbf{A}(t_k)$ is given by the Jacobian of the nonlinear system around the current state x_i, which has a closed-form solution,

$$\mathbf{A}(t_k) = \frac{\partial \mathbf{q}\left(\mathbf{x}(t_k)\right)}{\partial \mathbf{x}(t_k)}.$$

We initialize the filter state with a linear least-square regression on the first four measurements. While the filter could be initialized with only two measurements, we found that four yields a good tradeoff between latency and precision.

B.5.5 Trajectory Propagation

To propagate the trajectory, we transform the ball's position in image coordinates and its covariance to world coordinates. For the covariance transformation, we use a first-order approximation. We then use the dynamical model (B.8) to propagate the system in world coordinates. We define the world-coordinate origin to coincide with the quadrotor center. We compute a critical time t_{crit} at which the distance between the trajectory $\mathbf{r}(t)$ and the quadrotor is minimal and propagate the states up to that time. We find t_{crit} by

$$\frac{\partial}{\partial t}|\mathbf{r}(t_{\text{crit}})| \stackrel{!}{=} 0. \tag{B.13}$$

The more intuitive geometrical solution of this problem is illustrated in Figure B.7, which yields

$$\mathbf{r}(t_{\text{crit}}) \cdot \dot{\mathbf{r}}(t_{\text{crit}}) \stackrel{!}{=} 0, \tag{B.14}$$

which results in a cubic equation in t_{crit} that can be solved by, e.g., Cardano's method.

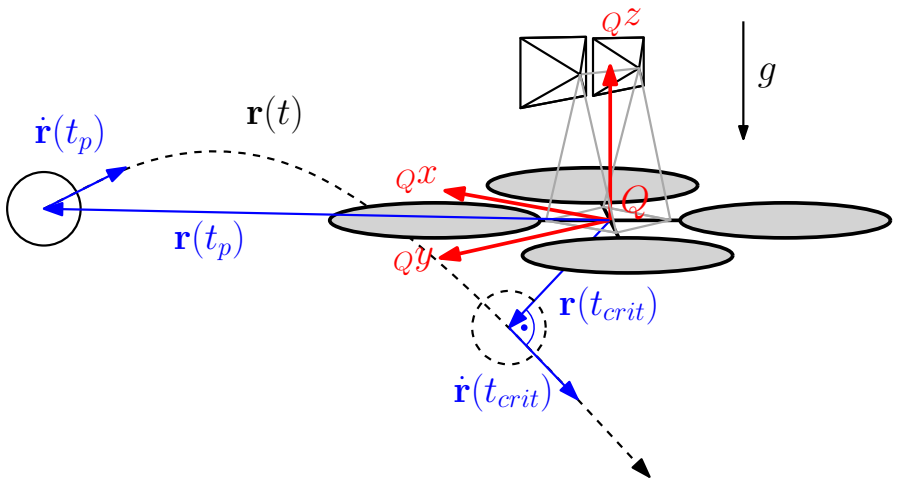


Figure B.7: Illustration of the geometrical solution to obtain the critical point $\mathbf{r}(t_{\text{crit}})$ on the ballistic trajectory $\mathbf{r}(t)$ defined by the vectors $\mathbf{r}(t_p)$ and $\dot{\mathbf{r}}(t_p)$. The critical point occurs at the time t_{crit}, when the position vector and its derivative are orthogonal.

B.5.6 Maneuver Decision

We define a spherical *safety zone* with radius r_{safe} around the quadrotor's origin. Since the trajectory prediction includes the expected value of the critical position and its uncertainty, a collision is predicted if the uncertainty ellipsoid around the critical position intersects with the safety zone.

If the distance-to-origin $\|\mathbf{r}(t_{\text{crit}})\|$ of the predicted critical position is smaller than r_{safe}, we expect a collision in any case. If it is larger, we need to consider the uncertainty of the prediction. The vector $\mathbf{r}_{\text{dist}}(t_{\text{crit}})$ (B.15) represents the distance from the expected critical position on the predicted trajectory to the closest point on the surface of the spherical safety zone,

$$\mathbf{r}_{\text{dist}}(t_{\text{crit}}) = \left(\frac{r_{\text{safe}}}{\|\mathbf{r}(t_{\text{crit}})\|} - 1 \right) \cdot \mathbf{r}(t_{\text{crit}}). \tag{B.15}$$

The covariance matrix $\mathbf{\Sigma}_r(t_{\text{crit}})$ can be interpreted as an ellipsoid that gives us information about directional uncertainty of the predicted critical position. This information is extracted with the Principal Component Analysis (PCA) of the covariance matrix

$$\mathbf{\Sigma}_r(t_{\text{crit}}) = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top \tag{B.16}$$

with

$$\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3] \quad \text{and} \quad \mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \lambda_3),$$

where the normalized eigenvectors $\mathbf{u}_i$ are the principal axes of the uncertainty ellipsoid and the square roots of the corresponding eigenvalues λ_i represent the standard deviations along the axes.

Therefore, we rotate $\mathbf{r}_{\text{dist}}$ to the PCA space U using $\mathbf{U}$ and scale it with $3\sqrt{\mathbf{\Lambda}}$, which corresponds to a probability of 99.73 %. We call this the *confidence vector*

$${}_U\mathbf{r}_{\text{conf}}(t_{\text{crit}}) = \left(3\sqrt{\mathbf{\Lambda}}\right)^{-1} \cdot \mathbf{U} \cdot \mathbf{r}_{\text{dist}}(t_{\text{crit}}). \tag{B.17}$$

The norm of the confidence vector then gives us an indication of the predicted critical position not entering the safety zone with a probability of 99.73 %. More precisely, we can rule out a collision with said probability, if the norm of the confidence vector is larger than 1.

Additionally, we set a threshold σ_λ for the largest eigenvalue that determines whether

we trust the prediction. I.e., we predict a collision if

$$\max(\mathbf{\Lambda}) \leq \sigma_\lambda \text{ and } \|\mathbf{r}(t_{\text{crit}})\| \leq r_{\text{safe}} \tag{B.18}$$

or

$$\max(\mathbf{\Lambda}) \leq \sigma_\lambda \text{ and } \|_U\mathbf{r}_{\text{conf}}(t_{\text{crit}})\| \leq 1. \tag{B.19}$$

B.6 Experiments

We first describe the experimental setup. Then, we evaluate the tracking performance and the effect of the EKF. We compare the measurements with a ground truth captured with a motion-capture system. Finally, we analyze the time margin between a collision is predicted and the time of collision.

B.6.1 Experimental Setup

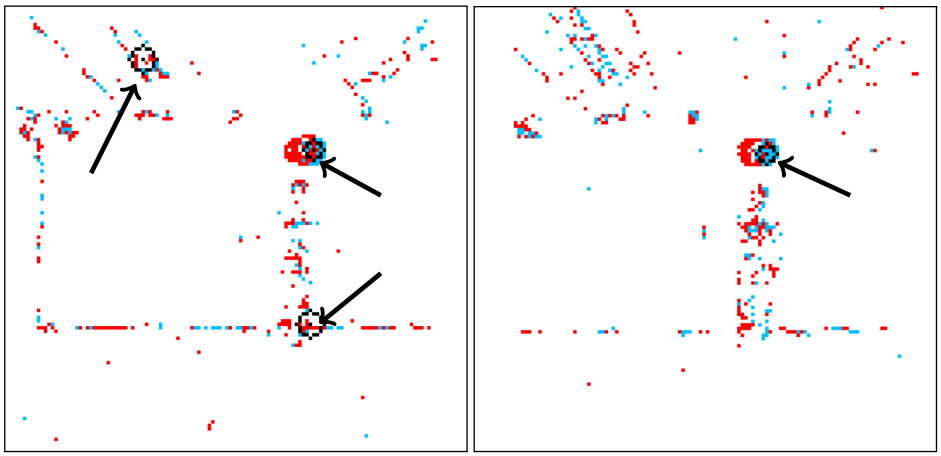
We mounted two DVS in a stereo setup on a quadrotor (see Fig. B.1a). Our quadrotor platform is described in detail in [45]. All sensing and computation for flying was performed onboard. We recorded the event streams from both DVS while hovering in vision-based flight that we later processed offboard. Then, we threw a ball towards the quadrotor from a distance of 6 m at speeds of about 10 m/s. The ball was secured with a leash that prevents a collision shortly before it would occur. We recorded ground truth for both the quadrotor and the ball using an OptiTrack motion-capture system.

B.6.2 Circle Tracking

Fig. B.8 shows a snapshot of the tracking. The three trackers on the left event stream in Fig. B.8a are filtered by the stereo-matching constraints. Since the trackers only roughly track the ball's position, a sub-pixel disparity refinement is used to improve the measurement.

B.6.3 EKF Performance

Figure B.9 shows the stereo measurements and the output of the EKF. The first four measurements between t_1 and t_2 are used for initialization of the filter. The last measurements at time t_3 are corrupted since the ball is no longer fully visible by the DVS. However, since these measurements do not pass a 3σ validation gate, they are not used to update the state and only the dynamical model is propagated.



(a) left DVS **(b)** right DVS

Figure B.8: Tracking of the ball during a throw. Red and blue points indicate events with positive and negative polarity, respectively. The active circle trackers are marked in black and highlighted with an arrow.

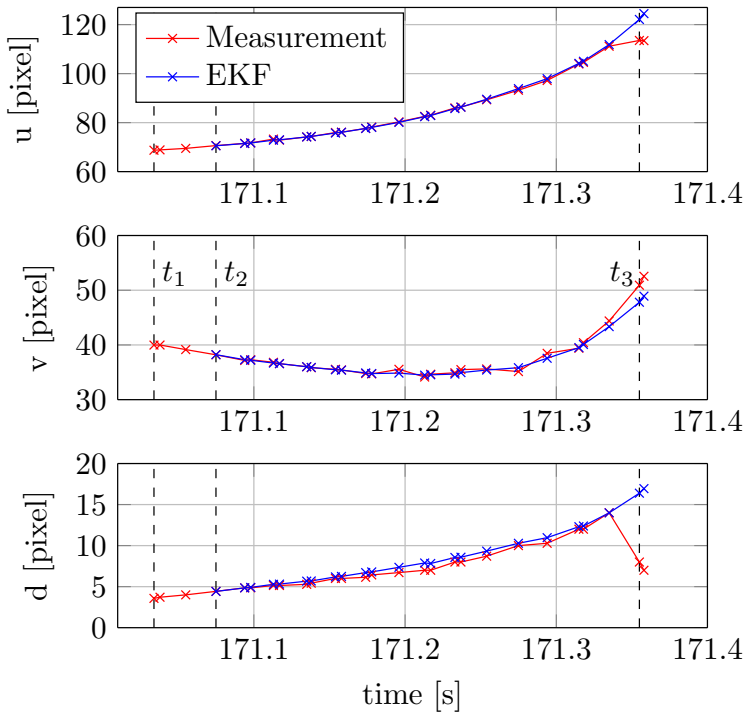


Figure B.9: Comparison of the measurement and the EKF estimate of the center of mass in image coordinates $[u, y, d]^\top$. The first measurement is obtained at t_1 and the EKF is initialized at t_2. The EKF recognizes corrupt measurement at t_3.

B.6.4 Comparison with Ground Truth

In Fig. B.10, the output of the EKF is compared to ground truth obtained from a motion-capture system. It also shows the triggering signal for the evasive maneuver.

The EKF is initialized at time t_2 and at time t_4, a triggering signal is sent. The elapsed time is $t_4 - t_2 = 15\,\text{ms}$. Since the ball was on a leash in the experiment, it stops before hitting the quadrotor. When we extrapolate its trajectory, a collision with the quadrotor would occur at time t_5. In this case, this yields a time of $t_5 - t_4 = 330\,\text{ms}$ for the quadrotor to escape before a collision would occur. Since the plots in Fig. B.10 are time-stamped by the same clock, they include the latency of the algorithm.

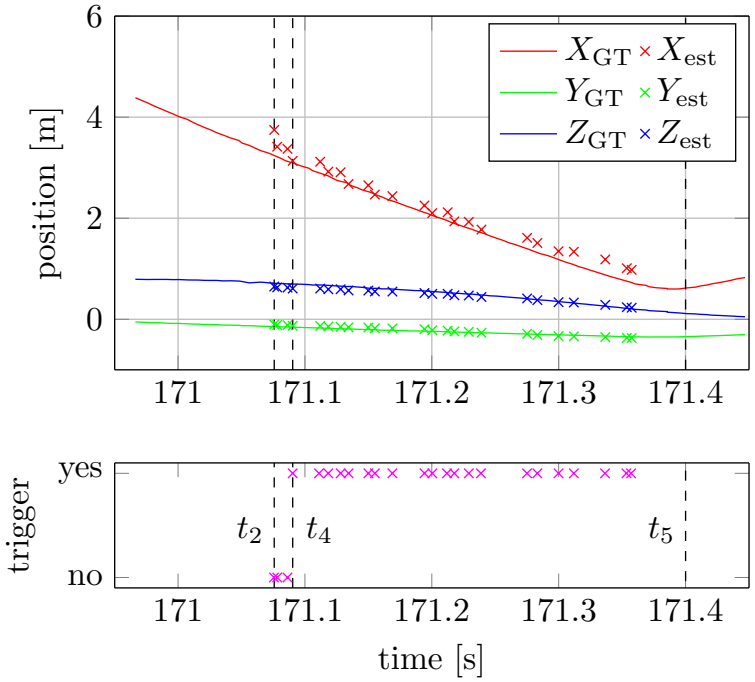


Figure B.10: Ground truth and center of mass estimate in quadrotor frame world coordinates. The EKF is initialized at t_2 and the collision prediction is made at t_4. The expected collision would take place at t_5. The evasive maneuver triggering signals are shown in magenta.

B.6.5 Time Margin for Evasive Maneuver

The time margin is the time between a collision is predicted and the time of collision (assuming no maneuver) We evaluate the time margin for the evasive maneuver for a series of 19 experiments. In 15 experiments, the ball was thrown at the quadrotor, which was correctly predicted 12 times (80 %), while it was not detected in 3 experiments. In the other 4 experiments, the ball missed the quadrotor. Our algorithm reported 3 times (75 %) that no collision will occur and did not detect the ball in one experiment.

We summarized the time margin for the 12 successfully predicted collisions in Fig. B.11. We estimated the time of collision by extrapolating the ball trajectory until it would hit the quadrotor. The algorithm ran in real-time on the recorded data, thus we incorporate also the processing time in our analysis. In most experiments, the time margin was

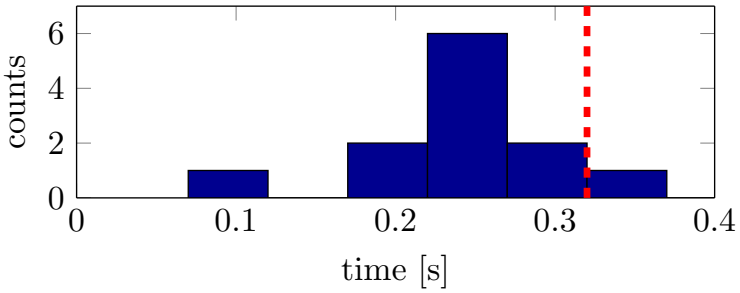


Figure B.11: Histogram of the escape time. The red dashed line indicates the 0.32 s escape time that is necessary for our quadrotor to escape the imminent collision with a free fall evasive maneuver (see Section B.7).

250 ms.

B.7 Conclusion

In this paper, we demonstrated a method to predict collisions of objects thrown at a quadrotor. We used two DVS in a stereo configuration. The ball was detected and tracked in both event streams. Using a set of geometric constraints, a rough stereo matching is obtained that is further refined by sub-pixel disparity estimation. These measurements are fused in an EKF that uses a mixed state to incorporate both the measurement noise in the image plane and the dynamical model of the ball's trajectory. We predict a collision if the 3σ-ellipsoid along the predicted trajectory intersects with a safety sphere around the quadrotor.

While the prediction of a collision is available within a reasonable time margin of 250 ms in most cases, our quadrotor platform currently does not allow to execute an aggressive evasive maneuver. In fact, due to the additional payload of two DVS and its sensor mount (which makes the quadrotor 40 % heavier), the quadrotor operates at its actuator limits (i.e., it can barely hover). The only possible evasive maneuver would be a free fall. However, to escape a safety sphere of $s = 0.5$ m using free fall, a time margin of at least $\sqrt{2s/g} = 0.32$ s is required, assuming no delays on the motor controllers and neglecting rotor dynamics. We are currently building a more powerful quadrotor platform to perform different evasive maneuvers in future work.

Acknowledgments

The authors wish to thank Matthias Faessler and Guillermo Gallego for their contributions to the sensor-latency measurements.

Appendix: Probabilistic Sensor-Latency Model

The total sensor latency Y is the sum of two independent random variables,

$$X_1 = \mathcal{N}(\mu, \sigma), \tag{B.20}$$
$$X_2 = \mathcal{U}(0, \Delta t), \tag{B.21}$$
$$Y = X_1 + X_2, \tag{B.22}$$

where X_1 models the Gaussian sensor latency and X_2 accounts for the uniform delay due to the fixed sensor frequency (cf. Sec. B.4.2). While Δt is known, we want to identify μ and σ. Since the two variables are independent, we can sum their expected values and variances,

$$\mathrm{E}[Y] = \mathrm{E}[X_1] + \mathrm{E}[X_2] = \mu + \frac{\Delta t}{2}, \tag{B.23}$$
$$\mathrm{Var}[Y] = \mathrm{Var}[X_1] + \mathrm{Var}[X_2] = \sigma^2 + \frac{\Delta t^2}{12}. \tag{B.24}$$

Given enough samples, we can compute $\mathrm{E}[Y]$ and $\mathrm{Var}[Y]$ from the measurements and compute the mean μ and variance σ^2 of the Gaussian as

$$\mu = \mathrm{E}[Y] - \frac{\Delta t}{2}, \tag{B.25}$$
$$\sigma^2 = \mathrm{Var}[Y] - \frac{\Delta t^2}{12}. \tag{B.26}$$

C Event-Camera Dataset and Simulator

Reprinted, with permission, from:

E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza. "The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM". in: *Int. J. Robot. Research* 36 (2 2017), pp. 142–149. DOI: 10.1177/0278364917691115

The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM

Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck and Davide Scaramuzza

Abstract — New vision sensors, such as the Dynamic and Active-pixel Vision sensor (DAVIS), incorporate a conventional global-shutter camera and an event-based sensor in the same pixel array. These sensors have great potential for high-speed robotics and computer vision because they allow us to combine the benefits of conventional cameras with those of event-based sensors: low latency, high temporal resolution, and very high dynamic range. However, new algorithms are required to exploit the sensor characteristics and cope with its unconventional output, which consists of a stream of asynchronous brightness changes (called "events") and synchronous grayscale frames. For this purpose, we present and release a collection of datasets captured with a DAVIS in a variety of synthetic and real environments, which we hope will motivate research on new algorithms for high-speed and high-dynamic-range robotics and computer-vision applications. In addition to global-shutter intensity images and asynchronous events, we provide inertial measurements and ground-truth camera poses from a motion-capture system. The latter allows comparing the pose accuracy of ego-motion estimation algorithms quantitatively. All the data are released both as standard text files and binary files (i.e., rosbag). This paper provides an overview of the available data and describes a simulator that we release open-source to create synthetic event-camera data.

C.1 Introduction

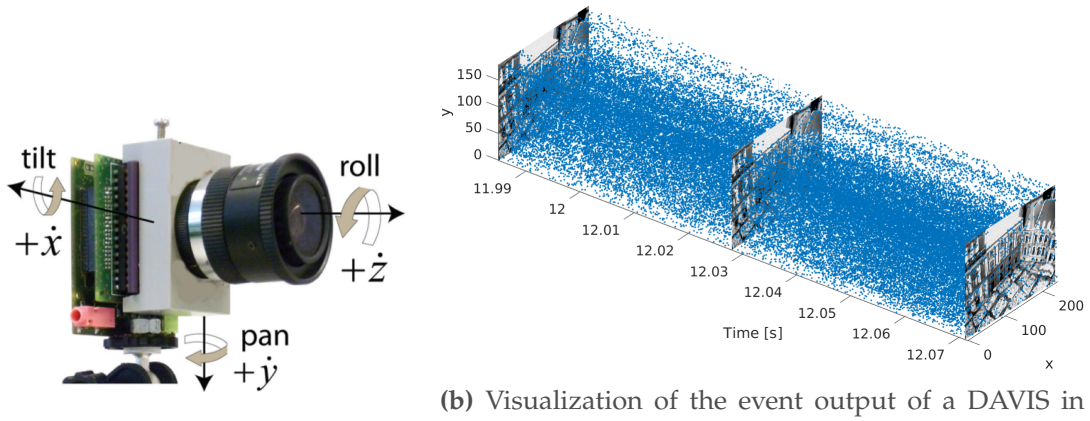
Over the past fifty years, computer-vision research has been devoted to standard, frame-based cameras (i.e., rolling or global shutter cameras) and only in the last few years cameras have been successfully used in commercial autonomous mobile robots, such as cars, drones, and vacuum cleaners, just to mention a few. Despite the recent progress, we believe that the advent of event-based cameras is about to revolutionize the robot sensing landscape. Indeed, the performance of a mobile robot in tasks, such as navigation, depends on the accuracy and latency of perception. The latency depends on the frequency of the sensor data plus the time it takes to process the data. It is typical in current robot-sensing pipelines to have latencies in the order of 50–200 ms or more, which puts a hard bound on the maximum agility of the platform. An event-based camera virtually eliminates the latency: data is transmitted using events, which have a latency in the order of *micro*-seconds. Another advantage of event-based cameras is their very high dynamic range (130 dB vs. 60 dB of standard cameras), which makes them ideal in scenes characterized by large illumination changes. Other key properties of event-based cameras are low-bandwidth, low-storage, and low-power requirements. All these properties enable the design of a new class of algorithms for high-speed and high-dynamic-range robotics, where standard cameras are typically not ideal because of motion blur, image saturation, and high latency. However, the way that event-based cameras convey the information is completely different from that of traditional sensors, so that a paradigm shift is needed to deal with them.

C.1.1 Related Datasets

There exist two recent datasets that also use the DAVIS: [127] and [7].

The first work is tailored for comparison of event-based optical flow estimation algorithms [127]. It contains both synthetic and real datasets under pure rotational (3 degrees of freedom (DOF)) motion on simple scenes with strong visual contrasts. Ground truth was acquired using the inertial measurement unit (IMU). In contrast, our datasets contain arbitrary, hand-held, 6-DOF motion in a variety of artificial and natural scenes with precise ground-truth camera poses from a motion-capture system.

A more similar work to ours is [7]. Their focus is to create a dataset that facilitates comparison of event-based and frame-based methods for 2D and 3D visual navigation tasks. To this end, a ground robot was equipped with a DAVIS and a Microsoft Kinect RGB-D sensor. The DAVIS was mounted on a pan-tilt unit, thus it could be excited in 5-DOF. The scene contains checkerboards, books, and a chair. Ground truth was acquired by the encoders of pan-tilt unit and the ground robot's wheel odometry, and is therefore subject to drift. In contrast, our dataset contains hand-held, 6-DOF motion (slow- and high-speed) on a variety of scenes with precise ground-truth camera poses



(a) The DAVIS sensor and axes definitions. Figure adapted from [37]

(b) Visualization of the event output of a DAVIS in space-time. Blue dots mark individual asynchronous events. The polarity of the events is not shown.

Figure C.1: The DAVIS camera and visualization of its output.

from a motion-capture system, which is not subject to drift.

C.2 The DAVIS Sensor

The Dynamic and Active-pixel Vision Sensor (DAVIS) [19] (see Fig. C.1a) is an event camera that transmits *events* in addition to frames. Events are pixel-level, relative-brightness changes that are detected in continuous time by specially-designed pixels[1]. The events are timestamped with *micro*-second resolution and transmitted asynchronously at the time they occur. Each event e is a tuple $\langle x, y, t, p \rangle$, where x, y are the pixel coordinates of the event, t is the timestamp of the event, and $p = \pm 1$ is the polarity of the event, which is the sign of the brightness change. This representation is sometimes also referred to as Address-Event Representation (AER). The DAVIS has a spatial resolution of 240×180 pixels. A visualization of the event output is shown in Fig. C.1b. Both the events and frames are generated by the same physical pixels, hence there is no spatial offset between the events and the frames.

Due to its low latency and high temporal resolution, both in the range of *micro*-seconds, event-based cameras are very promising sensors for high-speed mobile robot applications. Since event cameras are data-driven (only brightness *changes* are transmitted), no redundant data is transmitted. The required bandwidth thus depends on the motion speed and the type of scene. An additional advantage for robotic applications is the high dynamic range of 130 dB (compared to 60 dB of expensive computer-vision cameras), which allows both indoor and outdoor operation without changing parameters. Since all pixels are independent, very large contrast changes can also take place within the same scene.

[1] Video illustration: https://youtu.be/LauQ6LWTkxM

Over the course of the last seven years, several groups including ours have demonstrated the use of event-based sensors in a variety of tasks, such as SLAM in 2D [146] and 3D [70, 67, 124], optical flow [32, 9, 6], visual odometry [26], 6-DOF localization for high-speed robotics [102], line detection and localization [149], 3D reconstruction [123], image reconstruction and mosaicing [66, 125], orientation estimation [55], and continuous-time trajectory estimation [101].

However, all these methods were evaluated on different, specific datasets and, therefore, cannot be compared against each other. The datasets we propose here are tailored to allow comparison of pose tracking, visual odometry, and SLAM algorithms. Since event-based cameras, such as the DAVIS, are currently still expensive ($\sim 5,000$ USD), these data also allow researchers without equipment to use well-calibrated data for their research.

C.2.1 DAVIS IMU

In addition to the visual output (events and frames), the DAVIS includes an IMU that provides gyroscope and accelerometer data, thus enabling to design visual-inertial event-based algorithms. The DAVIS cameras has the IMU mounted directly behind and centered under the image sensor pixel array center, at a distance of about 3 mm from it, so that the IMU shares nearly the same position as the event sensor. The IMU axes are aligned with the camera axes (see Fig. C.1a). More specifically, the IMU is an InvenSense MPU-6150[2], which integrates a three-axis gyroscope that can measure in the range $\pm 2{,}000\,^\circ/\mathrm{s}$ and a three-axis accelerometer for the range $\pm 16g$. It integrates six 16-bit ADCs for digitizing the gyroscope and accelerometer outputs at 1 kHz sample rate.

C.3 DAVIS Simulator

Simulation offers a good baseline when working with new sensors, such as the DAVIS. Based on the operation principle of an ideal DAVIS pixel, we created a simulator that, given a virtual 3D scene and the trajectory of a moving DAVIS within it, generates the corresponding stream of events, intensity frames, and depth maps. We used the computer graphics software Blender[3] to generate thousands of rendered images along the specified trajectory, ensuring that the motion between consecutive images was smaller than 1/3 pixel. For each pixel, we keep track of the time of the last event triggered at that location. This map of timestamps (also called surface of active events [9]), combined with time interpolation of the rendered image intensities, allows determining brightness changes of predefined amount (given by the contrast threshold)

[2]IMU data sheet: https://store.invensense.com/ProductDetail/MPU6150-invensense/470090/
[3]https://www.blender.org/

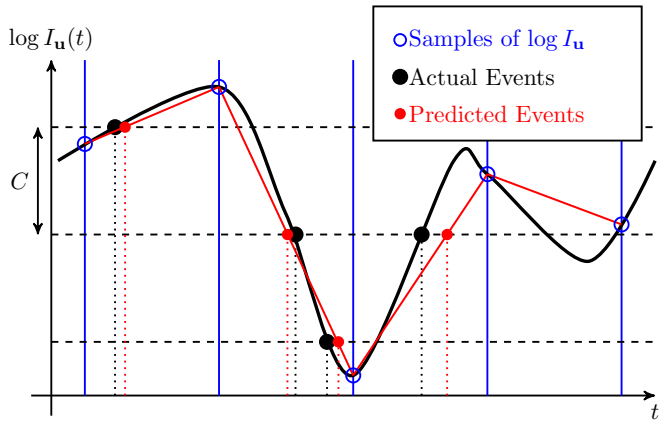


Figure C.2: DAVIS Simulator. Per-pixel event generation using piecewise linear time interpolation of the intensities given by the rendered images. For simplicity, images were rendered at a fixed rate.

in the time between images, thus effectively providing continuous timestamps, as if events were generated asynchronously. Time interpolation has an additional benefit: it solves the problem of having to generate millions of images for each second of a sequence, as it would have been required to deliver microsecond-resolution timestamps in the absence of interpolation.

More specifically, Fig. C.2 illustrates the operation of the simulator for a single pixel $\mathbf{u} = (x, y)^\top$. The continuous intensity signal at pixel $\mathbf{u}$, $\log I_{\mathbf{u}}(t)$ (black) is sampled at the times of the rendered images (blue markers). These samples are used to determine the times of the events: the data is linearly interpolated between consecutive samples and the crossings of the resulting lines (in red) with the levels given by multiples of the contrast threshold C (i.e., horizontal lines) specify the timestamps of the events (red dots). As it can be observed, this simple interpolation scheme allows for (i) higher resolution event time stamps than those of the rendered images, and (ii) the generation of multiple events between two samples if the corresponding intensity jump is larger than the contrast threshold.

The provided events are "perfect" measurements up to sampling and quantization; under this condition, an image $\hat{I}(\mathbf{u}; t)$ can be reconstructed from the event stream at any point in time t by accumulating events $e_k = \langle \mathbf{u}_k, t_k, p_k \rangle$ according to

$$\log \hat{I}(\mathbf{u}; t) = \log I(\mathbf{u}; 0) + \sum_{0 < t_k \leq t} p_k \, C \, \delta(\mathbf{u} - \mathbf{u}_k) \delta(t - t_k),$$

where $I(\mathbf{u}; 0)$ is the rendered image at time $t = 0$ and δ selects the pixel to be updated on every event (pixel $\mathbf{u}_k$ of $\hat{I}$ is updated at time t_k). We used this scheme to check that the reconstructed image agreed with the rendered image at several points in time; specifically, the per-pixel intensity error was confined to the quantization interval

$(-C, C)$.

Event generation operates on brightness pixels, which are computed from the rendered color images using the ITU-R Recommendation BT.601[4] for luma, i.e., according to formula $Y = 0.299R + 0.587G + 0.114B$, with RGB channels in linear color space to better resemble the operation of the DAVIS.

Because realistic event noise is extremely difficult to model due to the complex behavior of event sensors with respect to their bias settings and other factors, the provided simulation datasets do not include event noise. Nevertheless, the simulator, and the datasets created with it, are a useful tool for prototyping new event-based algorithms. Our implementation is available as open-source software.[5]

C.4 Datasets

In this section, we describe the datasets that we provide. The datasets contain:

- the asynchronous event stream,
- intensity images at about 24 Hz,
- inertial measurements (3-axis gyroscope and 3-axis accelerometer) at 1 kHz,
- ground-truth camera poses from a motion-capture system[6] with sub-millimeter precision at 200 Hz (for the indoor datasets),
- the intrinsic camera matrix.

All information comes with precise timestamps. For datasets that were captured outside the motion-capture system (e.g., in an office or outdoors), no ground truth is provided. Some datasets were collected using a motorized linear slider and ground truth was collected using the slider's position. Due to vibrations induced by the slider motor, the very noisy IMU data was not recorded.

C.4.1 Data Format

The datasets are provided in standard text form that is described here. For convenience, they can also be downloaded as binary rosbag files (the details are on the website). The format of the text files is described in Table C.1.

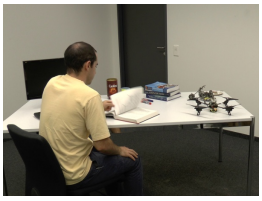
[4] https://www.itu.int/rec/R-REC-BT.601
[5] https://github.com/uzh-rpg/rpg_davis_simulator
[6] We use an OptiTrack system from NaturalPoint.

(a) Shapes

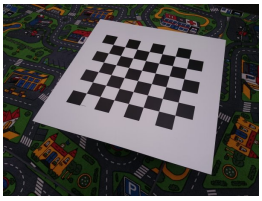
(b) Wall Poster

(c) Boxes

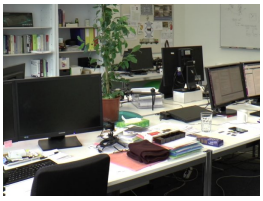
(d) Outdoors



(e) Dynamic



(f) Calibration

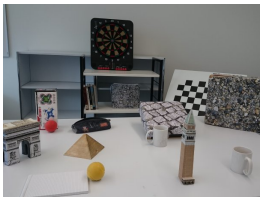


(g) Office

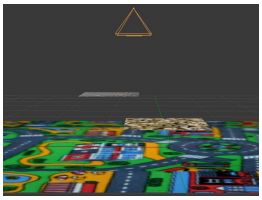
(h) Urban

(i) Motorized linear slider

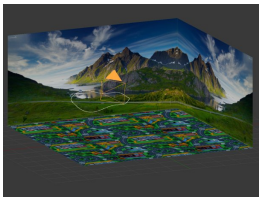
(j) Motorized slider (HDR)



(k) Motorized slider with objects



(l) Synthetic: 3 planes



(m) Synthetic: 3 walls

Figure C.3: Dataset scenes

File	Description	Line Content
`events.txt`	One event per line	`timestamp x y polarity`
`images.txt`	One image reference per line	`timestamp filename`
`images/00000000.png`	Images referenced from `images.txt`	
`imu.txt`	One measurement per line	`timestamp ax ay az gx gy gz`
`groundtruth.txt`	One ground truth measurements per line	`timestamp px py pz qx qy qz qw`
`calib.txt`	Camera parameters	`fx fy cx cy k1 k2 p1 p2 k3`

Table C.1: Description of Dataset Format

The ground-truth pose is with respect to the (arbitrary) motion-capture origin that has the z-axis gravity-aligned (pointing upwards). The orientation is provided as a unit quaternion $\mathbf{q} = (q_x, q_y, q_z, q_w)^\top$, where q_w and $\mathbf{q}_v = (q_x, q_y, q_z)^\top$ are the scalar and vector components, respectively. This convention was proposed as a standard by JPL [21].

All values are reported in SI units. While the timestamps were originally recorded as POSIX, we subtracted the lowest timestamp as offset such that all datasets start at zero. This helps to avoid numerical difficulties when dealing with microsecond resolution timestamps of the events.

Images are provided as PNG files. The list of all images and their timestamps is provided in a separate file. The typical framerate is 24 Hz, but it varies with the exposure time.

The IMU axes are aligned with the optical coordinate frame (i.e., the positive z-axis is identical to the optical axis and so are the x- and y-axes).

C.4.2 List of Datasets

The provided datasets are summarized in Table C.2 and Fig. C.3. All the datasets contain increasing speeds, different scenes, and varying degrees of freedom[7]: for the `shapes`, `poster`, and `boxes` datasets, the motion first starts with excitation of each single degree of freedom separately; then combined and faster excitations are performed. This leads to increasing difficulty and a higher event rate over time.

In the high-dynamic-range (HDR) sequences (`hdr_poster`, `hdr_boxes`, and `slider_hdr`), a spotlight was used to create large intrascene contrasts. For `hdr_poster`, we measured 80 lx and 2,400 lx in the dark and bright areas, respectively.

The `outdoors` datasets were acquired in an urban environment both walking and running. While no ground truth is available, we returned precisely to the same location after a large loop.

The `dynamic` datasets were collected in a mock-up office environment viewed by the motion-capture system, with a moving person first sitting at a desk, then moving around.

A `calibration` dataset is also available, for instance in case the user wishes to use a different camera model or different methods for hand-eye calibration. The dimensions of the calibration pattern (a checkerboard) are 6×9 tiles of 40 mm. For the lower half of the table, different settings (lenses, focus, etc.) were used. Thus, while we provide the intrinsic calibration, no calibration datasets are available.

The `slider_close`, `slider_far`, `slider_hdr_close`, and `slider_hdr_far` datasets were recorded with a motorized linear slider parallel to a textured wall at 23.1 cm, 58.4 cm, 23.2 cm, and 58.4 cm, respectively.

[7] The DAVIS was moved by hand, the dominant motion is described.

Name	Motion	Scene	GT	T [s]	TS [m/s]	RS [°/s]	NE [-]
shapes_rotation	Rotation, incr. speed	Fig. C.3a	yes	59.8	0.83	730	23,126,288
shapes_translation	Translation, incr. speed	Fig. C.3a	yes	59.7	2.60	271	17,363,976
shapes_6dof	6 DOF, incr. speed	Fig. C.3a	yes	59.7	2.35	715	17,962,477
poster_rotation	Rotation, incr. speed	Fig. C.3b	yes	59.8	0.84	884	169,350,136
poster_translation	Translation, incr. speed	Fig. C.3b	yes	59.8	2.58	240	100,033,286
poster_6dof	6 DOF, incr. speed	Fig. C.3b	yes	59.8	2.51	937	133,464,530
boxes_rotation	Rotation, incr. speed	Fig. C.3c	yes	59.8	0.85	669	185,688,947
boxes_translation	Translation, incr. speed	Fig. C.3c	yes	59.8	3.43	319	112,388,307
boxes_6dof	6 DOF, incr. speed	Fig. C.3c	yes	59.8	3.84	509	133,085,511
hdr_poster	6 DOF, incr. speed	Fig. C.3b	yes	59.8	2.28	597	102,910,720
hdr_boxes	6 DOF, incr. speed	Fig. C.3c	yes	59.8	2.94	592	118,499,744
outdoors_walking	6 DOF, walking	Fig. C.3d	no†	133.4	n/a	n/a	64,517,638
outdoors_running	6 DOF, running	Fig. C.3d	no†	87.6	n/a	n/a	98,572,164
dynamic_rotation	Rotation, incr. speed	Fig. C.3e	yes	59.8	0.45	542	71,324,510
dynamic_translation	Translation, incr. speed	Fig. C.3e	yes	59.8	1.86	227	35,809,924
dynamic_6dof	6 DOF, incr. speed	Fig. C.3e	yes	59.7	2.91	627	57,174,637
calibration	6 DOF, slow	Fig. C.3f	yes	59.8	0.32	67	21,340,629
office_zigzag	6-DOF, zigzag, slow	Fig. C.3g	no	10.9	n/a	n/a	7,735,308
office_spiral	6-DOF, spiral, slow	Fig. C.3g	no	11.2	n/a	n/a	6,254,774
urban	Linear, slow	Fig. C.3h	no	10.7	n/a	n/a	5,359,539
slider_close	Linear, const, speed	Fig. C.3i	yes*	6.5	0.16	0	4,032,668
slider_far	Linear, const, speed	Fig. C.3i	yes*	6.4	0.16	0	3,442,683
slider_hdr_close	Linear, const. speed	Fig. C.3j	yes*	6.5	0.16	0	3,337,787
slider_hdr_far	Linear, const. speed	Fig. C.3j	yes*	6.5	0.16	0	2,509,582
slider_depth	Linear, const. speed	Fig. C.3k	yes*	3.4	0.32	0	1,078,541
simulation_3planes	Translation, circle	Fig. C.3l	yes#	2.0	0.63	0	6,870,278
simulation_3walls	6 DOF	Fig. C.3m	yes#	2.0	5.31	109	4,104,833

Table C.2: List of Datasets. Note that the calibration dataset only applies to the upper half of the table. The other datasets use different lenses and calibrations. GT: Ground truth. T: Duration. TS: Maximum translation speed. RS: Maximum rotational speed. NE: Number of events. †Same start and end pose after a large loop. *Ground truth from motorized linear slider. No IMU data due to vibrations. #Simulated DAVIS using Blender. No IMU data included.

For the datasets, we applied two different sets of biases (parameters) for the DAVIS, as listed in Table C.3. The first set, labeled "indoors", was used in all datasets but `outdoors_walking`, `outdoors_running`, and `urban`, where the set "outdoors" was applied. For the simulated datasets, we used a contrast threshold of $\pm 15\,\%$ and $\pm 20\,\%$ for the `simulation_3planes` and `simulation_3walls`, respectively.

For the simulated scenes, we also provide the 3D world model in Blender (cf. Fig. C.3l and C.3m). In addition to the intensity images and events, these datasets include a depth map for each image frame at 40 Hz, encoded as 32-bit floating-point values (in the OpenEXR data format).

C.5 Calibration

First, we calibrated the DAVIS intrinsically using a checkerboard pattern. Then, we computed the hand-eye calibration that we applied to the subsequent dataset recordings so that the ground-truth poses that we provide are those of the event camera (i.e., the

Bias	Indoors		Outdoors	
	Coarse	Fine	Coarse	Fine
DiffBn	2	39	4	39
OFFBn	1	62	4	0
ONBn	4	200	6	200
PrBp	3	72	2	58
PrSFBp	3	96	1	33
RefrBp	3	52	4	25

Table C.3: List of biases applied to the DAVIS. The DAVIS uses two stages of biases, coarse and fine, which we report here.

"eye"), not those of the motion-capture trackable (i.e., the "hand") attached to the camera. We also included a calibration dataset in case a different camera model or improved hand-eye calibration method is required.

C.5.1 Intrinsic Camera Calibration

We used the standard pinhole camera model with radial-tangential distortion using the implementation of ROS and OpenCV[8]. We used three radial distortion coefficients (k_1, k_2, and $k_3 = 0$) and two for tangential distortion (p_1 and p_2). The distortion coefficients are listed in `calib.txt` in the same order as in OpenCV. We provide a dataset for post-calibration in case that another method is preferred.

C.5.2 Hand-Eye Calibration

For the indoor datasets, we provide accurate and high-frequency (200 Hz) pose data from a motion-capture system. However, the coordinate frame used by the motion-capture system is different from the optical coordinate frame of the DAVIS. Thus, we performed a hand-eye calibration before acquiring the datasets. Fig. C.4 shows the coordinate frames and transformations used to solve the hand-eye calibration problem. The frames are those of the world W, the hand H, the camera E (Fig. C.1a), and the checkerboard C. For the transformations, Fig. C.4 shows both the compact standard notation of hand-eye calibration problems and a more explicit one: the Euclidean transformation $\mathtt{T}_{ba}$ (4×4 homogeneous matrix representation) maps points from frame a to frame b according to $\mathbf{P}_b = \mathtt{T}_{ba}\mathbf{P}_a$.

More specifically, we first use a linear algorithm [140] to provide an initial solution of the hand-eye calibration problem $\{\mathtt{A}_i\mathtt{X} = \mathtt{X}\mathtt{B}_i\}_{i=1}^{N}$, where $\mathtt{A}_i \leftrightarrow \mathtt{B}_i$ are N correspondences of relative hand-hand ($\mathtt{A}_i := \mathtt{T}_{H_k H_j}$) and eye-eye ($\mathtt{B}_i := \mathtt{T}_{E_k E_j}$) poses at different times (j and k), respectively, and $\mathtt{X} := \mathtt{T}_{HE}$ is the unknown eye-to-hand transformation.

[8] http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

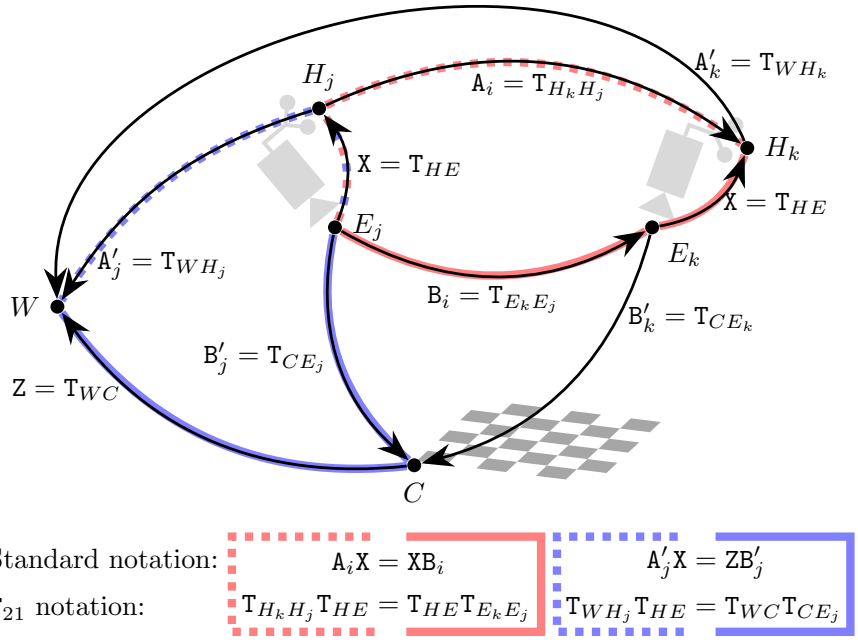


Figure C.4: Hand-eye calibration. Coordinate frames and transformations involved in case of the hand-eye device at two different positions (j and k). The red loop between two stations of the hand-eye device is used in the first type of hand-eye calibration problems, of the form $\mathtt{A}_i\mathtt{X} = \mathtt{X}\mathtt{B}_i$, and the blue loop is used in the second type of hand-eye calibration problems, of the form $\mathtt{A}'_j\mathtt{X} = \mathtt{Z}\mathtt{B}'_j$. We use a combination of both approaches to solve for the constant, hand-eye calibration transform $\mathtt{X}$.

Then, using the second formulation of hand-eye calibration problems, of the form $\{\mathtt{A}'_j\mathtt{X} = \mathtt{Z}\mathtt{B}'_j\}_{j=1}^{N+1}$, where $\mathtt{A}'_j := \mathtt{T}_{WH_j}$ and $\mathtt{B}'_j := \mathtt{T}_{CE_j}$ are the hand-to-motion-capture and eye-to-checkerboard transformations for the j-th pose, respectively, we refined $\mathtt{T}_{HE}$ by jointly estimating the hand-eye $\mathtt{X}$ and robot-world $\mathtt{Z} := \mathtt{T}_{WC}$ (i.e., motion-capture–checkerboard) transformations that minimize the reprojection error in the image plane:

$$\min_{\mathtt{X},\mathtt{Z}} \sum_{mn} d^2\Big(\mathbf{x}_{mn}, \hat{\mathbf{x}}_{mn}(\mathtt{X}, \mathtt{Z}; \mathtt{A}'_m, \mathbf{P}_n, \mathtt{K})\Big),$$

where $d^2(\mathbf{x}_{mn}, \hat{\mathbf{x}}_{mn})$ is the squared Euclidean distance between the measured projection $\mathbf{x}_{mn}$ of the n-th checkerboard corner $\mathbf{P}_n$ on the m-th camera and the predicted corner $\hat{\mathbf{x}}_{mn} = \mathbf{f}(\hat{\mathtt{B}}'_m; \mathbf{P}_n, \mathtt{K})$, which is a function of the intrinsic camera parameters $\mathtt{K}$ and the extrinsic parameters $\hat{\mathtt{B}}'_m := \mathtt{Z}^{-1}\mathtt{A}'_m\mathtt{X}$ predicted using the motion-capture data. This non-linear least-squares problem is solved iteratively using the Gauss-Newton method. The initial value of $\mathtt{Z}$ is given by $\mathtt{Z} = \mathtt{A}'_1\mathtt{X}\mathtt{B}'^{-1}_1$, with $\mathtt{X}$ provided by the above-mentioned linear algorithm. We included a dataset for post-calibration in case another method is preferred.

The ground-truth pose gives the position and orientation of the event camera with respect to the world (i.e., the motion-capture system). Hence, it already incorporates the computed hand-eye transformation. That is, while the motion-capture system outputs $\mathtt{T}_{WH_j}$, we apply the hand-eye calibration $\mathtt{T}_{HE} \equiv \mathtt{T}_{H_jE_j}\ \forall j$ and directly report $\mathtt{T}_{WE_j} = \mathtt{T}_{WH_j}\mathtt{T}_{H_jE_j}$ as ground-truth pose.

C.6 Known Issues

C.6.1 Clock Drift and Offset

The clocks from motion-capture system and the DAVIS are not hardware-synchronized. We observed clock drift of about 2 ms/min. To counteract the clock drift, we reset the clocks before each dataset recording. Since all datasets are rather short (in the order of 1 min), the effect of drift is negligible. A small, dataset-dependent offset between the DAVIS and motion-capture timestamps is present since the timestamps were reset in software.

D Event Lifetime

E. Mueggler, C. Forster, N. Baumli, G. Gallego, and D. Scaramuzza. "Lifetime Estimation of Events from Dynamic Vision Sensors". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2015, pp. 4874–4881. DOI: 10.1109/ICRA.2015.7139876

Lifetime Estimation of Events from Dynamic Vision Sensors

Elias Mueggler, Christian Forster, Nathan Baumli, Guillermo Gallego and

Davide Scaramuzza

Abstract — We propose an algorithm to estimate the "lifetime" of events from retinal cameras, such as a Dynamic Vision Sensor (DVS). Unlike standard CMOS cameras, a DVS only transmits pixel-level brightness changes ("events") at the time they occur with *micro*-second resolution. Due to its low latency and sparse output, this sensor is very promising for high-speed mobile robotic applications. We develop an algorithm that augments each event with its lifetime, which is computed from the event's velocity on the image plane. The generated stream of augmented events gives a continuous representation of events in time, hence enabling the design of new algorithms that outperform those based on the accumulation of events over fixed, artificially-chosen time intervals. A direct application of this augmented stream is the construction of sharp gradient (edge-like) images at any time instant. We successfully demonstrate our method in different scenarios, including high-speed quadrotor flips, and compare it to standard visualization methods.

D.1 Introduction

D.1.1 Motivation

Event-based (retinal) vision sensors [36], such as the Dynamic Vision Sensor [77], offer great potential for robotic applications: since only pixel-level brightness *changes* are transmitted, less bandwidth is required and less data must be processed. In addition, these changes are transmitted at the time they occur with minimal latency, which is in

the order of a few *micro*-seconds. Due to the asynchronous nature of these changes, they are called *events*.

However, since an event stream is fundamentally different from video streams of standard CMOS cameras, new algorithms are required to deal with this data. Event-based adaptations of iterative closest points [108] and optical flow [10, 9] have been proposed. Recently, event-based visual odometry [26, 66], tracking [145, 102], and Simultaneous Localization And Mapping (SLAM) [146] algorithms were also presented. The design goal of such algorithms is that each incoming event can asynchronously change the estimated state, thus preserving the event-based nature of the sensor.

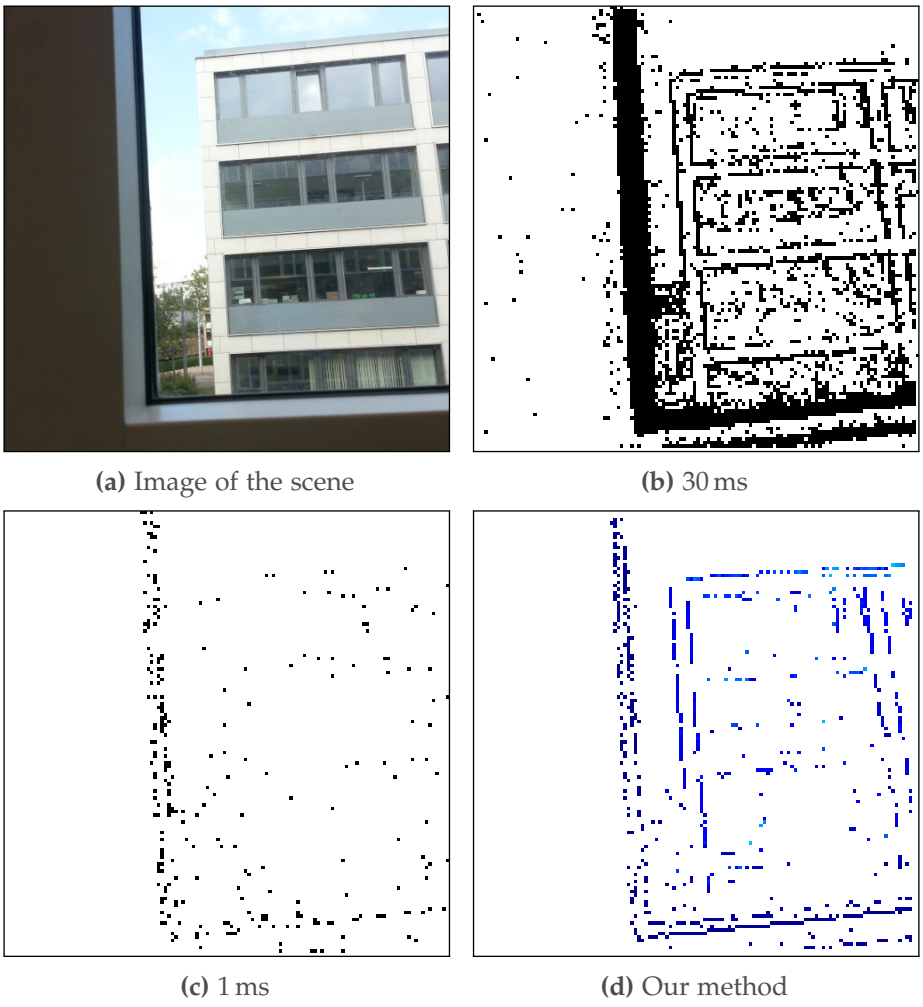
While all of these algorithms implicitly buffer a certain number of past events, we propose to explicitly model the set of active events. We consider an event active as long as the brightness gradient causing this event is visible by the pixel. The estimation of such a set of active events has several applications, such as the generation of sharp gradient images at any point in time, clustering of events for tracking of multiple objects, etc. Here, we focus on the first one (see Fig. D.1). This also allows applying standard computer-vision algorithms on these images without modification.

D.1.2 Related Work

Due to its low latency and low bandwidth, the DVS [77] is a promising sensor for robotic systems with limited computational power and short time constants. An impressive demonstration of these capabilities was presented in [31]. Using two DVS, the authors implemented a pencil-balancing system on a highly-reactive platform free to move on a plane. A robotic goalkeeper with a reaction time of 3 ms was presented in [34]. More recently, robot localization was demonstrated using a DVS during high-speed maneuvers [102], where rotational speeds of up to 1,200 °/s were measured during quadrotor flips.

Standard computer-vision algorithms cannot be applied directly to the output of event-based vision sensors, since they do not provide grayscale intensity images. A straightforward workaround is to generate such intensity images by accumulating events over a fixed time interval and then apply standard frame-based algorithms. An event-to-frame converter was presented in [69] and tested on two conventional stereo-vision algorithms. Another example of DVS event accumulation was shown in [130], where events were accumulated in artificial time slots of 5–50 ms and used in stereo vision for tracking moving objects. In both cases, the event-to-frame conversion was a time-consuming process that introduced some latency and, therefore, the asynchronous data delivery and high temporal resolution of the DVS was not used very efficiently.

In [108], the events in a sliding window of fixed duration were selected as input of an Iterative Closest Point (ICP) algorithm that was used to guide a micro gripper to



(a) Image of the scene **(b)** 30 ms

(c) 1 ms **(d)** Our method

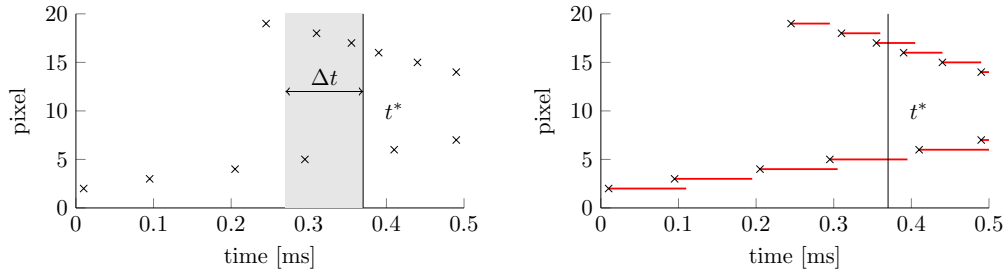
Figure D.1: The Dynamic Vision Sensor (DVS) is moved in front of a window frame diagonally, from bottom-left to top-right a. Since the window frame is much closer than the buildings, its apparent motion is significantly larger. Thus, if we use a fixed accumulation interval, the images will either be blurred, if the interval is too long b, or some structures will not be clearly visible, if the interval is too short c. Our method estimates the lifetime of each event independently and displays the event for that period of time d.

grasp an object with a mean update rate of 4 kHz. In this particular setup, such a fixed duration could be chosen for all the pixels of the DVS, because the gripper was moving at almost constant speed parallel to the image plane. In a general configuration, however, such a time interval does not exist.

D.1.3 Contributions and Outline

In this paper, we present a method to augment data streams from event-based cameras with their *lifetime* and the velocity of each event, while simultaneously filtering noise.

Our method is based on the event-based optical flow [10, 9] to estimate the velocity of an event from where we can estimate its lifetime. As a direct application, this method allows rendering *sharp* gradient images at *any* point in time, as illustrated in Fig. D.2.



(a) Raw event stream **(b)** Event stream augmented with lifetime

Figure D.2: In this illustration, we consider a single pixel row of a DVS, which observes two edges moving at different speeds (cf. Fig. D.4). The events are marked with crosses. To visualize the events at time t^*, the active events at that time are plotted. If a fixed accumulation interval Δt is chosen a, some regions become blurred (the upper, fast edge is represented with two events) or others are not complete (if we chose a shorter interval, the slow edge would not be considered). Our method b assigns a *lifetime* to each event (shown in red), thus the active events at t^* are known.

In contrast to previous algorithms, our method does not depend on a temporal window $[t - \Delta t, t + \Delta t]$ around the event time t. Thus, we eliminate both a tuning parameter (Δt) and its corresponding latency (our method uses only *past* events). Our method is also robust against noise because we use RANSAC [48] and a regularization term. The output of the method can be used to apply standard computer-vision algorithms to the output of event-based cameras.

The remainder of the paper is organized as follows. In Section D.2, we characterize the Dynamic Vision Sensor (DVS). The developed algorithm to calculate the lifetime of an event is described in Section D.3 and evaluated in Section D.4.

D.2 Dynamic Vision Sensor

Standard CMOS cameras send full frames at fixed frame rates. On the other hand, event-based (retinal) cameras such as the DVS [77] have independent pixels that generate spike events at local relative brightness changes in continuous time. These events are timestamped and transmitted asynchronously at the time they occur using sophisticated digital circuitry. Each event e is a tuple $\langle x, y, t, p \rangle$, where x, y are the pixel coordinates of the event, t is the timestamp of the event, and $p \in \{-1, +1\}$ is the polarity of the event, which is the sign of the brightness change. This representation is sometimes also referred to as Address-Events Representation (AER). The DVS has a resolution of 128×128 pixels and is connected via USB. A visualization of the output of the DVS is

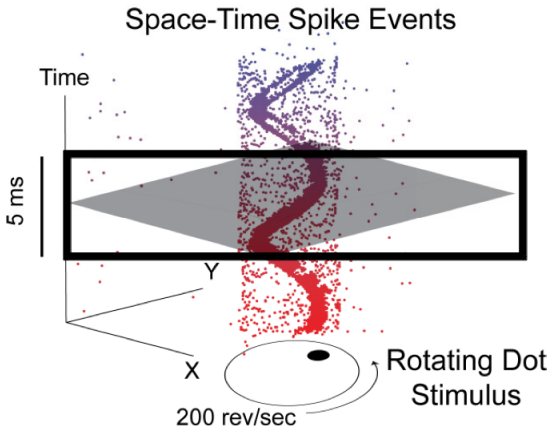


Figure D.3: Visualization of the output of a DVS looking at a rotating dot. Colored dots mark individual events. The polarity of the events is not shown. Events that are not part of the spiral are caused by sensor noise. Figure adapted from [82].

shown in Fig. D.3.

Due to its low latency and high temporal resolution, both in the range of *micro*-seconds, the DVS is a very promising sensor for high-speed mobile robot applications. Since the data stream from the DVS is sparse (only *changes* are reported), the bandwidth and computational load are low. An additional advantage for robotic applications is the DVS' high dynamic range of 120 dB (compared to 60 dB of expensive computer-vision cameras), which allows both indoor and outdoor operation without changing parameters. Since all pixels are independent, these contrasts can also take place within the same scene.

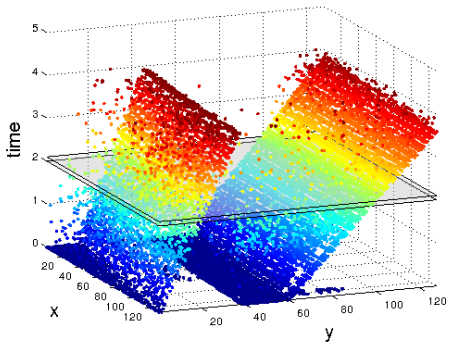
D.3 Algorithm

In this section, we devise our algorithm to estimate the lifetime of each incoming event. The basic idea is to determine each event's velocity $\mathbf{v} = (v_x, v_y)^\top$ on the image plane and use this information to calculate the time interval that this event is considered active. The lifetime τ indicates how long it will take for the brightness gradient at the current event location to trigger a new event in a neighboring pixel. We assign zero lifetime to noise events, $\tau = 0$.

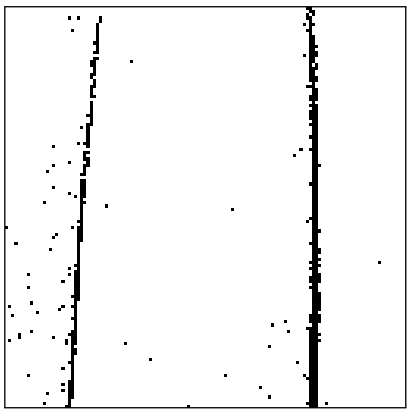
Our algorithm augments the stream of events $\langle x, y, t, p \rangle$ with the the lifetime τ and the event's velocity $\mathbf{v}$,

$$\langle x, y, t, p \rangle \mapsto \langle x, y, t, p, \tau, v_x, v_y \rangle. \tag{D.1}$$

The velocity of the event $\mathbf{v}$ is computed using event-based visual flow, which is



(a) Surface of Active Events with a slice shown in b.



(b) All events between $t = 2\,\text{s}$ and $2.1\,\text{s}$.

Figure D.4: Surface of Active Events (SAE) of two lines moving to the right. The left line moves slower than the right one. Events caused by sensor noise are visible as isolated dots. A 100 ms slice starting at $t = 2\,\text{s}$ is shown in b, which corresponds to the naive method that accumulates events over a fixed time interval (here, 100 ms). While the slow line appears sharp, the fast line is several pixels wide, which corresponds to motion blur.

estimated based on the method introduced in [10, 9]. We first present our adaptation of the event-based visual flow and the computation of the lifetime for each event. Then, we detail the local plane-fitting algorithm including outlier rejection and regularization.

D.3.1 Event-Based Visual Flow and Lifetime

The Surface of Active Events (SAE) is defined in the three-dimensional spatio-temporal domain that is composed of the two-dimensional sensor frame and an additional dimension representing time [1]. Each incoming event generates or updates a point on the surface, such that, for each pixel position on the image plane, the time value of the surface is equal to the timestamp of the last event at this position. The SAE is given by the map $\Sigma_e : \mathbb{R}^2 \to \mathbb{R}$, $t = \Sigma_e(\mathbf{p})$, where $\mathbf{p} = (x, y)^\top$. In space-time, a point of the SAE is represented by the 3-vector $S(\mathbf{p}) = (x, y, \Sigma_e(x, y))^\top$. In this sense, $\Sigma_e(\mathbf{p})$ represents the SAE as an "elevation map".

Figure D.4a shows the SAE of real data recorded with the DVS in the spatio-temporal domain. The recorded sequence contains two lines moving at different speeds, hence the different slopes. Sensor noise is clearly visible as isolated dots. Figure D.4b shows the corresponding visualization of a 100 ms slice. The latter corresponds to what we refer to as the *naive* method that accumulates events over a fixed time interval.

The planar approximation of the SAE at an event's location $\mathbf{p}$ is given by the first order

Taylor expansion

$$S(\mathbf{p}+\Delta\mathbf{p}) \approx S(\mathbf{p}) + \Big(S_x(\mathbf{p}), S_y(\mathbf{p})\Big)\Delta\mathbf{p}, \tag{D.2}$$

where $S_x = \frac{\partial S}{\partial x} = (1, 0, \frac{\partial \Sigma_e}{\partial x})^\top$, $S_y = \frac{\partial S}{\partial y} = (0, 1, \frac{\partial \Sigma_e}{\partial y})^\top$ are the first partial derivatives of S, representing vectors in the tangent space to S.

As illustrated in Fig. D.5, we define the lifetime of the event at $(\mathbf{p}, t)$ as the first order approximation of the maximum temporal increment of S for a displacement $\|\Delta\mathbf{p}\| = 1$ pixel:

$$\tau(\mathbf{p}) = \max \Delta t \quad \text{subject to} \quad \|\Delta\mathbf{p}\| = 1, \tag{D.3}$$

where $\Delta t = \langle S(\mathbf{p}+\Delta\mathbf{p}) - S(\mathbf{p}), \mathbf{e}_3\rangle$, $\mathbf{e}_3 = (0,0,1)^\top$ is the direction of the time axis and $\langle\cdot,\cdot\rangle$ is the standard inner product in $\mathbb{R}^n$.

The lifetime τ, therefore, indicates the maximum amount of time before the brightness gradient at the current event location will trigger a new event in a neighboring pixel.

Substituting (D.2) and the expressions for S_x, S_y in (D.3) yields $\Delta t = \mathbf{e}_3^\top \Big(S_x(\mathbf{p}), S_y(\mathbf{p})\Big)\Delta\mathbf{p} = \langle \nabla\Sigma_e(\mathbf{p}), \Delta\mathbf{p}\rangle$, with $\nabla\Sigma_e(\mathbf{p}) = \Big(\frac{\partial\Sigma_e}{\partial x}(\mathbf{p}), \frac{\partial\Sigma_e}{\partial y}(\mathbf{p})\Big)^\top$. Hence we arrive at the equivalent definition

$$\tau(\mathbf{p}) = \max \langle \nabla\Sigma_e(\mathbf{p}), \Delta\mathbf{p}\rangle \text{ subject to } \|\Delta\mathbf{p}\| = 1. \tag{D.4}$$

Since t is an increasing function, Σ_e is a monotonically increasing function of $\mathbf{p}$, thus it has nonzero gradient at any point $\mathbf{p}$, and $\nabla\Sigma_e(\mathbf{p})$ is related to the velocities describing the visual flow (see [9]) according to

$$\nabla\Sigma_e(\mathbf{p}) = \Big(v_x^{-1}(\mathbf{p}), v_y^{-1}(\mathbf{p})\Big)^\top. \tag{D.5}$$

The local planar approximation is equivalent to assuming constant velocities v_x and v_y.

The maximum (D.4) is achieved for the unit vector $\Delta\mathbf{p} = \nabla\Sigma_e(\mathbf{p})/\|\nabla\Sigma_e(\mathbf{p})\|$ (see Fig. D.5). Hence,

$$\tau(\mathbf{p}) = \|\nabla\Sigma_e(\mathbf{p})\| = \sqrt{v_x^{-2} + v_y^{-2}}. \tag{D.6}$$

Next, we give a formula for τ in terms of the normal to the surface S at $\mathbf{p}$ (i.e., the normal of the tangent plane), $\mathbf{n}(\mathbf{p}) = (n_1, n_2, n_3)^\top$, which is assumed to be known by fitting a plane to the data (this step will be described next in Section D.3.2). We may further assume that $n_3 > 0$ since Σ_e is a monotonically increasing function. The

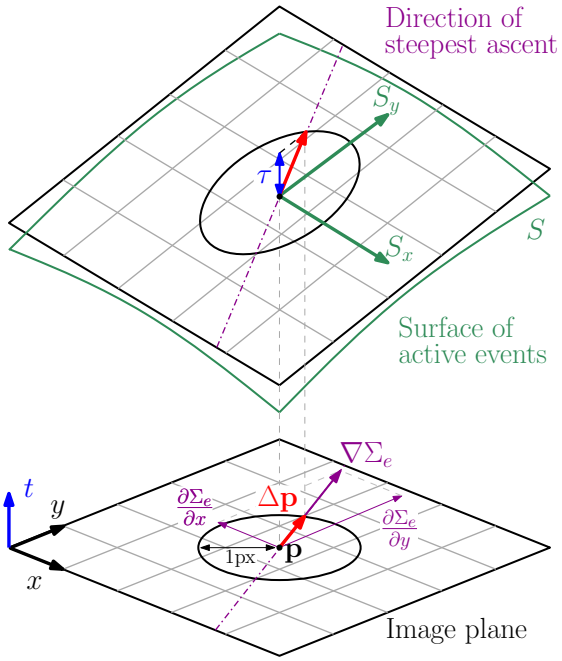


Figure D.5: Visualization of the lifetime τ: the maximum time increment Δt of the planar approximation to the Surface of Active Events (SAE) for a displacement of $\|\Delta\mathbf{p}\| = 1$ pixel. Both the optimal unit displacement $\Delta\mathbf{p}$ and τ are directly related to $\nabla\Sigma_e$, as summarized in (D.6).

normal is given by $\mathbf{n}(\mathbf{p}) \propto S_x(\mathbf{p}) \times S_y(\mathbf{p}) = (-(\nabla\Sigma_e(\mathbf{p}))^\top, 1)^\top$. Substituting (D.5) gives $\mathbf{n}(\mathbf{p}) \propto (-v_x^{-1}, -v_y^{-1}, 1)^\top$ in terms of the motion velocity and, identifying coordinates with $\mathbf{n}(\mathbf{p}) = (n_1, n_2, n_3)^\top$, we obtain $-v_x^{-1} = n_1/n_3$ and $-v_y^{-1} = n_2/n_3$, which finally implies

$$\tau(\mathbf{p}) = \sqrt{v_x^{-2} + v_y^{-2}} = \frac{1}{n_3}\sqrt{n_1^2 + n_2^2}. \tag{D.7}$$

D.3.2 Local Plane-fitting Algorithm

Our plane-fitting algorithm is based on [9], where all events in an $N \times N \times 2\Delta t$ window, centered around the current event, in the spatio-temporal domain are used to estimate the local plane. However, their algorithm has two undesirable properties: first, it introduces a tuning parameter (Δt) that limits the slowest detectable gradients (the slope of the plane). Second, events from the future are included, which translates to introducing a Δt latency. To overcome these issues, we only use *past* events in our estimation. Since we assume local smoothness, we can only use half of the events of the $N \times N$ window around the current event. This is illustrated in Fig. D.6.

To robustly fit the plane, we use the RANSAC algorithm [48]. We compute a candidate

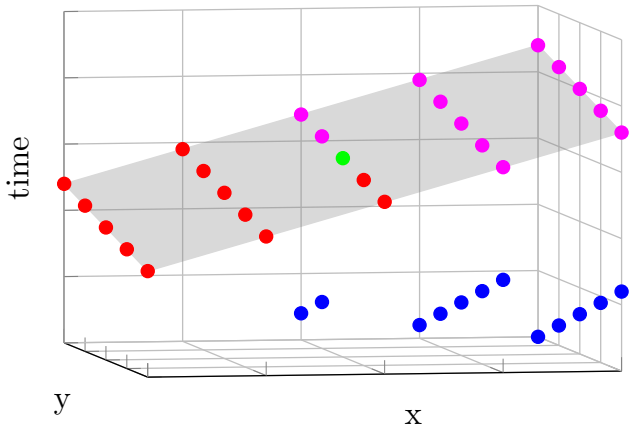


Figure D.6: When a new event (green) arrives, the SAE on a 5×5 patch around it includes the red and blue events. The events in red correspond to the brightness gradient that moves over this patch. The events in blue correspond to another gradient that moved over this patch previously. The events in magenta are future events to which we do not have access. However, under the local planar assumption, the magenta events will lie on the same plane (gray) as the red events. To avoid latency in our algorithm, we only use the newer half of events from the SAE (red events) to estimate the plane, while the blue events are not considered (see Section D.3.2).

plane using the new event and two additional past events that were chosen randomly. We then check all other past events whether they support the candidate plane. A past event is considered an inlier, if its point-to-plane distance is below the inlier threshold μ. The second tuning parameter of the RANSAC algorithm is the estimated percentage of outliers ϵ, which can further be used to estimate the necessary number of iterations. If less than m inliers are found, the event is considered as noise and its lifetime is set to zero. We compute m as a function of half the events in the window of size N and the percentage of outliers ϵ,

$$m = \underbrace{(1-\epsilon)}_{\substack{\text{inlier} \\ \text{ratio}}} \underbrace{N^2/2}_{\substack{\text{maximum} \\ \text{support}}} . \tag{D.8}$$

Both parameters have to be empirically tuned and they vary for different scenes and DVS settings. We found $\mu = 10^{-4}$ and $\epsilon = 0.4$ to yield good results. Note that we split incoming events by their polarity, i.e., we run our algorithm separately for the positive and negative events, and combine the output of both for the final result.

Plane Fitting

Let $\mathbf{A}$ be the matrix of the n inliers obtained by the RANSAC algorithm,

$$\mathbf{A} = \begin{pmatrix} x_1 & y_1 & t_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & t_n \end{pmatrix}, \tag{D.9}$$

where here $x_i, y_i, t_i, i \in \{1, \dots n\}$ are local coordinates relative to the current event.

The ordinary least-squares solution of the plane normal $\mathbf{n}_{LS}$ is given by

$$\mathbf{n}_{LS} = \arg\min_{\mathbf{n}} \|\mathbf{A}\mathbf{n} - \mathbf{b}\|^2, \tag{D.10}$$

where $\mathbf{b} = (1 \cdots 1)^\top$.

Regularization

To refine the estimate of the plane normal, we predict future events using the local constant-velocity assumption, which serves as regularization. Using the estimated velocity for a new event, we predict the time $\hat{t}$ for all neighboring pixels at which an event should occur. We then compare the time an event actually occurs with the predicted time. We use this difference as a measure of how much we trust the previously fitted plane. The absolute error

$$\Delta t_{\text{err}} = |t_i - \hat{t}_i| \tag{D.11}$$

between the predicted time $\hat{t}_i$ and the actual time t_i is, therefore, used for an error-dependent regularization weight $\lambda(\Delta t_{\text{err}})$. A regularized plane $\mathbf{n}_R$ is computed,

$$\mathbf{n}_R = \arg\min_{\mathbf{n}} \left(\|\mathbf{A}\mathbf{n} - \mathbf{b}\|^2 + \lambda(\Delta t_{\text{err}}) \|\mathbf{n} - \hat{\mathbf{n}}_i\|^2 \right), \tag{D.12}$$

where $\hat{\mathbf{n}}_i$ is the predicted plane normal.

The value of the error-dependent regularization weight $\lambda(\Delta t_{\text{err}})$ in (D.12) gives an indication about the preference of the prior information, e.g., for small prediction errors, the prior information is considered reliable and is therefore weighted stronger. Therefore, $\lambda(\Delta t_{\text{err}})$ should be big for small errors. An exponential approach is chosen to satisfy this condition. To enforce general smoothness on the motion, a constant value can be added to the exponential function. For the experiments described in this paper, the following function is used:

$$\lambda(\Delta t_{\text{err}}) = 9 + 100 \exp(-0.005 \Delta t_{\text{err}}). \tag{D.13}$$

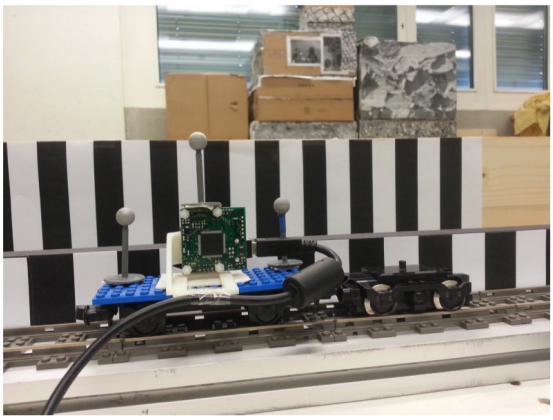


Figure D.7: The DVS is mounted on a train cart to achieve constant linear velocity. The scene is divided in three parts, which are at different depths: two line patterns (at 0.1 m and 0.2 m) and the background consisting of boxes and windows (at 5 m).

Edge Thinning

If a gradient is accelerating and thus violating the constant-velocity assumption, the lifetime will be overestimated. Therefore, two neighboring events in the direction of motion will be active at the same time, causing a similar effect as motion blur. A simple solution to this problem is to use the velocity information of the new event to reset the lifetime of the neighboring pixel in negative velocity direction. This technique effectively suppresses motion blur caused by accelerating gradients.

D.4 Experimental Evaluation

We evaluate our algorithm using four different experiments, going from controlled environments to urban settings. We visually compare the output of our method with that of the naive method that accumulates events over a fixed time interval.

D.4.1 Experiment 1: Line Pattern at Constant Velocity

Experimental Setup

The first experiment investigates the response to straight lines at different depths when the DVS moves parallel to the pattern at a constant velocity (see Fig. D.7). To enforce constant linear velocity, the DVS is mounted on a train cart. Two boards with distinct vertical black and white bars are installed in front of the DVS at different distances. In the DVS, both boards as well as part of the background containing cardboard boxes and windows are visible.

Results

Figure D.8 shows the event-stream visualization using the naive method with a fixed accumulation interval of 1 ms a and 30 ms b along with our algorithm both without c and with d regularization. Both intervals are not suitable for this setup, resulting in motion blur (30 ms) or hardly recognizable structure (1 ms). Our algorithm detects the lines and estimates their velocities coherently. The colors in the visualization correspond to the lifetime. The slow apparent background motion, visible in Fig. D.8b, is only partially captured by our method due to the bad signal-to-noise ratio for slow apparent motion. Qualitative comparison of the output with and without regularization shows that both perform similarly, with regularization performing slightly better when it comes to noise suppression. In this case of pure translation, the lifetime is proportional to the inverse depth of the scene. Hence, another application of the estimation of the lifetime of the events is the recovery of the structure (i.e., depth) of the scene.

Figure D.9 shows the fraction of cumulated predictions plotted against their absolute error. For instance, using $N = 5$ more than 90 % of all predictions have a smaller error than 10 ms. This quantitative evaluation for different window sizes does not show a significant difference between the algorithm with or without regularization for window sizes of $N = 5$ and $N = 7$. The effect of the regularization becomes much clearer when looking at the distribution of the estimated lifetimes (Fig. D.10). There is a stronger segregation of the two main expected lifetimes as a result of the regularization.

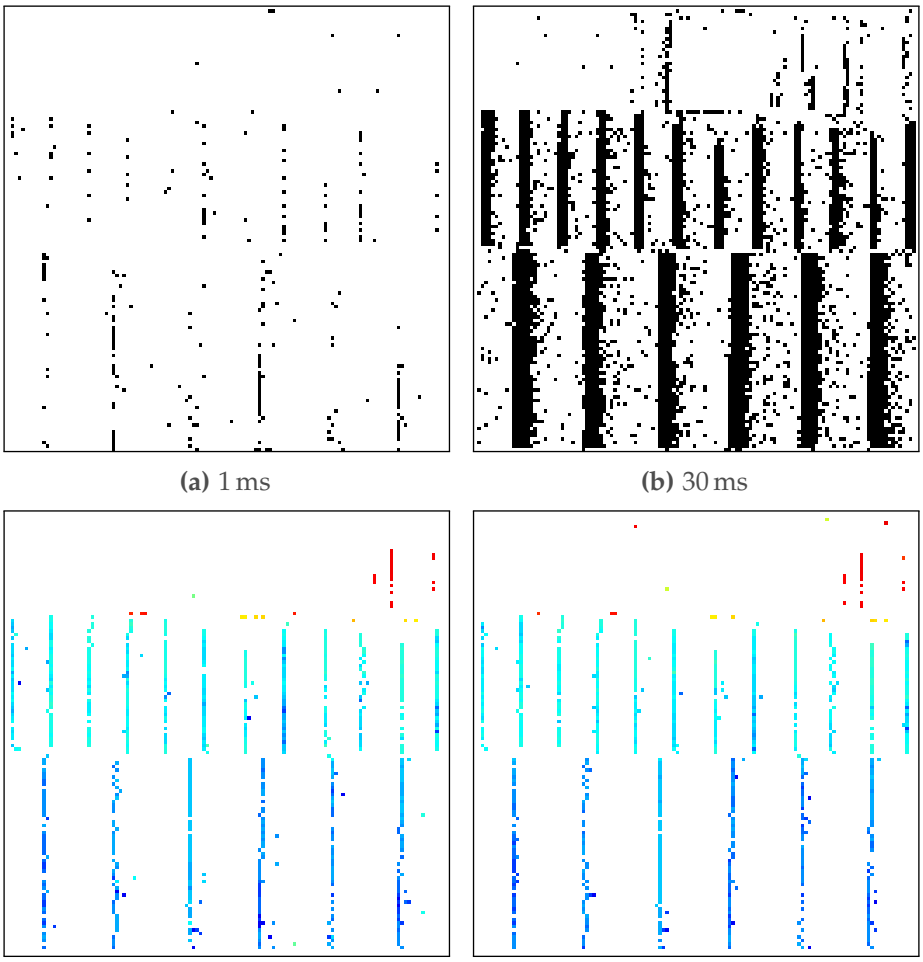
D.4.2 Experiment 2: Complex Patterns at Constant Velocity

Experimental Setup

The second experiment investigates the response to complex patterns. We used the same train cart to move the DVS at constant velocity parallel to a complex pattern (see Fig. D.11a). In this experiment, however, the entire pattern has constant distance to the DVS.

Results

Figure D.11b shows the output of our algorithm for Experiment 2. The silhouette of "Garfield" is clearly visible even though most edges are curved. Many details are preserved well. However, some details are too small to be captured by the low resolution of the DVS, which is 128×128 pixels. Horizontal edges are not visible, as they are parallel to the apparent motion and, therefore, do not trigger any events. In this setup, a well-tuned fixed lifetime would achieve similar results, but without noise suppression.



(a) 1 ms **(b)** 30 ms

(c) Without regularization **(d)** With regularization

Figure D.8: Experiment 1: DVS moves at constant velocity in front of a striped pattern. The experimental setup is shown in Fig. D.7. Accumulating events over a fixed time interval results in either unclear structures a or motion blur b. Since the apparent motion changes over the image space, no fixed interval exists that can render sharp images. Our method delivers sharp images as well as suppresses noise. Long and short lifetimes are depicted in red and blue, respectively.

D.4.3 Experiment 3: Quadrotor Flips

Experimental Setup

In this experiment, we mounted the DVS on a quadrotor in a front-looking configuration. The quadrotor first hovers in front of a black square attached to a white wall. It then performs a flip around the optical axis of the DVS and settles down to hover condition again. Figure D.12 shows the quadrotor performing a flip, when it reaches rotational speeds of up to 1,200 °/s.

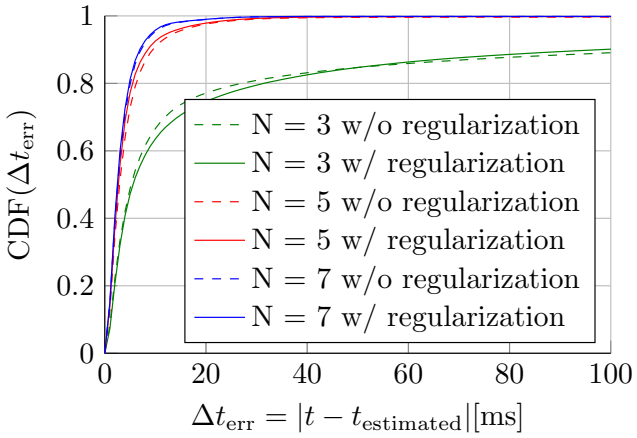
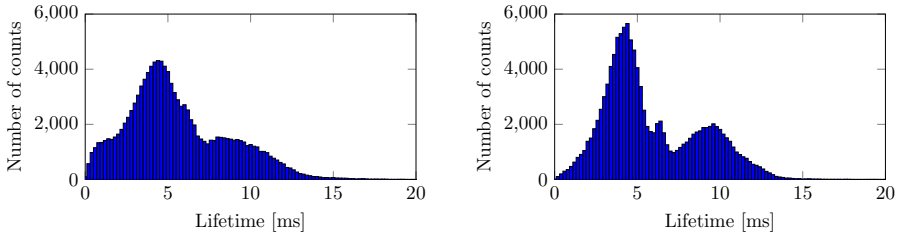


Figure D.9: Prediction error analysis of Experiment 1 for different window sizes both with and without regularization.



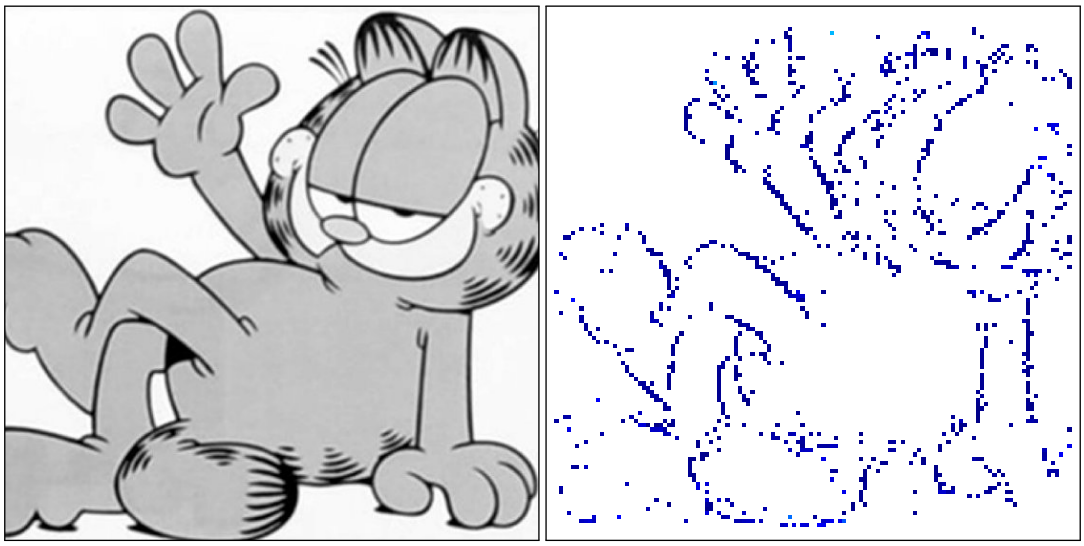
(a) Without regularization **(b)** With regularization

Figure D.10: Lifetime histogram of Experiment 1 for $N = 5$: events with an assigned lifetime within 0.2 ms are binned. The two peaks correspond to the close and far lines in the scene (cf. Fig. D.7). Note that with regularization b, the two peaks are much sharper than without a.

Results

Figures D.13 and D.14 compare the output of both the naive and proposed methods at two different time instances during the experiment. Figure D.13 shows the output during hovering, while Fig. D.14 corresponds to the flip.

During hovering, a fixed accumulation interval of 1 ms hardly captures any structure (Fig. D.13a), while 30 ms yields an almost sharp image (Fig. D.13b). During the flip, an interval of 1 ms yields a sharp image of the square (Fig. D.14a), while an interval of 30 ms causes heavy motion blur (Fig. D.14b). Thus, choosing such a accumulation interval results in a trade-off between completeness of the image and motion blur, which is visible as "thickening". In contrast, our method provides a sharp image in both situations, showing the applicability and adaptability of the algorithm to varying velocities in both rotational and translational motion.



(a) Image of the scene **(b)** Output of our algorithm

Figure D.11: Experiment 2: DVS moves at constant velocity parallel to the image shown in a. The output of our algorithm captures many details b. However, fine details are not preserved due to the low resolution of the DVS (128 × 128 pixels).

D.4.4 Experiment 4: Urban Environment

Figure D.1 shows an experiment in an urban environment. The scene constists of a window frame at a close distance with office buildings outside (cf. Fig. D.1a). While the naive method either misses elements in the scene (the buildings, see Fig. D.1c) or causes motion blur (the window frame, Fig. D.1b), our method performs well in capturing both fast and slow edges (Fig. D.1d).

D.5 Conclusion

We developed a method to augment the stream of events from a retinal camera with a measure of the *lifetime* of each event. Such a measure is defined based on the visual flow in the image plane and is computed based on a local planar approximation of the surface of active events. To this end, we designed an event-based, robust plane fitting algorithm with minimum latency (by considering only past events in the neighborhood of the current event) and optional regularization. In contrast to the previous work of visual-flow estimation, we did not rely on any temporal window or the use of future events.

The generated stream of augmented events opens up new possibilities to design other event-based algorithms that operate on a continuous-time representation of the events. This significantly departs from the algorithmic paradigm of event accumulation over

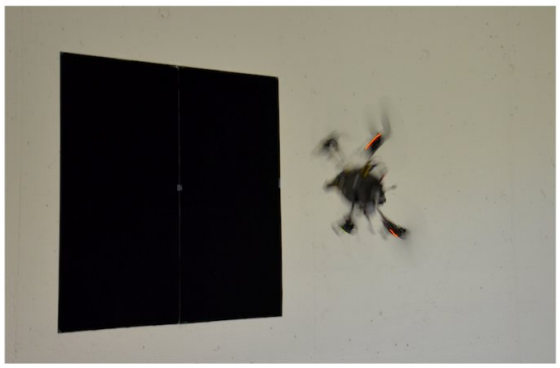
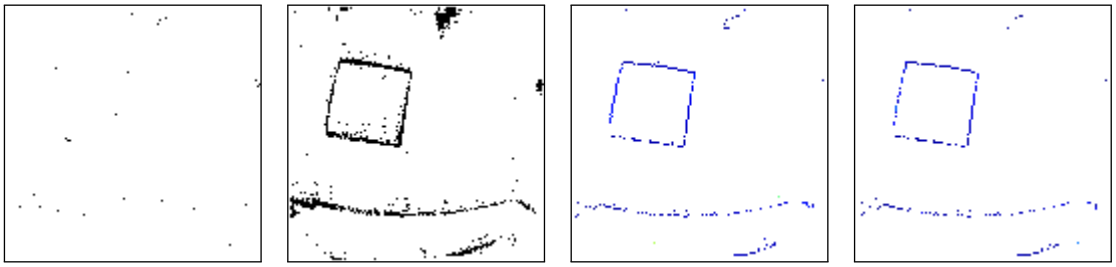


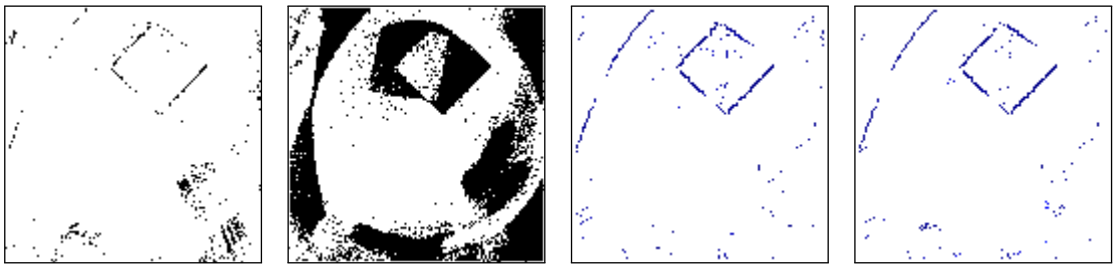
Figure D.12: Setup of Experiment 3: a quadrotor equipped with a front-looking DVS performs a flip in front of a square pattern. Rotational speeds are measured to be as high as 1,200 °/s during the flip.



(a) 1 ms **(b)** 30 ms **(c)** Without regularization **(d)** With regularization

Figure D.13: Experiment 3: Quadrotor during *hovering*. The experimental setup is shown in Fig. D.12. A fixed accumulation interval of 1 ms captures hardly any structure a, while an interval of 30 ms yields a sharp image b. Our method produces even sharper images and reduces noise c, d.

artificially-chosen intervals of fixed duration at discrete times, which suffer from the "completeness – motion blur" trade-off. We demonstrated the usefulness of our method with several experiments in a visualization application: the rendering of sharp gradient images at any time instant. Additionally, we included datasets acquired by a DVS onboard a quadrotor during agile maneuvers. Our method outperformed that of fixed event-accumulation interval, which implicitly assigns the same lifetime to all events, since it can cope with scenes containing structures at different apparent velocities. In addition, it is able to filter a significant amount of events caused by sensor noise. Our method performs well despite the low resolution of the DVS (128×128 pixels).



(a) 1 ms **(b)** 30 ms **(c)** Without regularization **(d)** With regularization

Figure D.14: Experiment 3: Quadrotor during the *flip*. The experimental setup is shown in Fig. D.12. A fixed accumulation interval of 1 ms yields a sharp image a, while an interval of 30 ms produces heavy motion blur b. Our method produces sharp images and reduces noise c, d.

E Event-based Feature Detection

Reprinted, with permission, from:

E. Mueggler, C. Bartolozzi, and D. Scaramuzza. “Fast Event-based Corner Detection”. In: *British Machine Vis. Conf. (BMVC)*. 2017

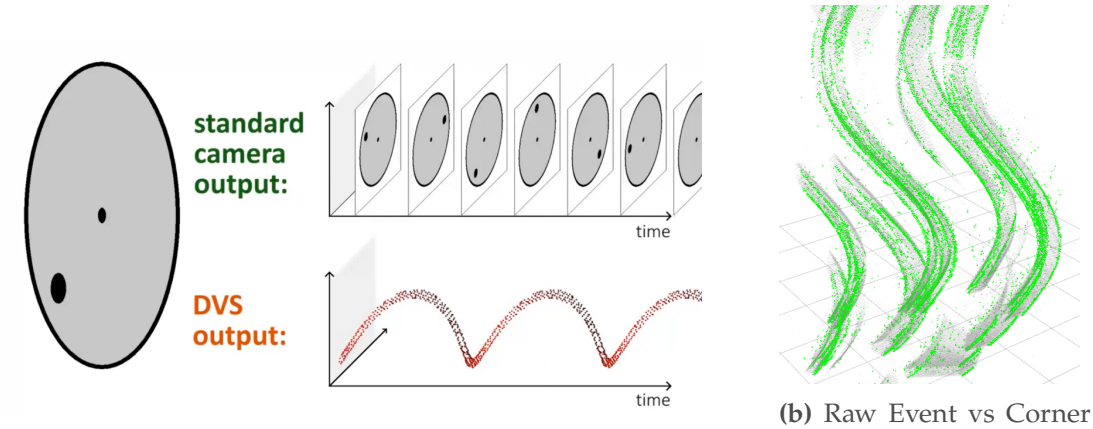
Fast Event-based Feature Detection

Elias Mueggler, Chiara Bartolozzi and Davide Scaramuzza

Abstract — Event cameras offer many advantages over standard frame-based cameras, such as low latency, high temporal resolution, and a high dynamic range. They respond to pixel-level brightness changes and, therefore, provide a sparse output. However, in textured scenes with rapid motion, millions of events are generated per second. Therefore, state-of-the-art event-based algorithms either require massive parallel computation (e.g., a GPU) or depart from the event-based processing paradigm. Inspired by frame-based pre-processing techniques that reduce an image to a set of features, which are typically the input to higher-level algorithms, we propose a method to reduce an event stream to a *corner event* stream. Our goal is twofold: extract relevant tracking information (corners do not suffer from the aperture problem) and decrease the event rate for later processing stages. Our event-based corner detector is very efficient due to its design principle, which consists of working on the Surface of Active Events (a map with the timestamp of the latest event at each pixel) using only comparison operations. Our method asynchronously processes event by event with very low latency. Our implementation is capable of processing millions of events per second on a single core (less than a *micro*-second per event) and reduces the event rate by a factor of 10 to 20.

E.1 Introduction

Event cameras offer great potential for virtual reality and robotics to overcome the challenges of latency, dynamic range, and high speed. Inspired by the human eye, these cameras respond to local, pixel-level brightness changes at the time they occur. These changes, called “events”, are transmitted asynchronously and timestamped



(a) Standard vs Event Camera

(b) Raw Event vs Corner Event Stream

Figure E.1: (a): Comparison of the output of a standard frame-based and an event camera when facing a rotating disk with a black dot. The standard camera outputs frames at a fixed rate, thus sending redundant information when no motion is present in the scene. Standard cameras also suffer from motion blur during rapid motion. Event cameras instead respond to pixel-level brightness changes with microsecond latency. Therefore, they do not suffer from motion blur and do not report anything when everything is at rest. An animated version of this figure can be found here: https://youtu.be/LauQ6LWTkxM. (b): The output of our method is a corner event stream (green), which is here overlaid on the raw event stream (gray) in space-time (time going upwards).

with *micro*-second precision. A comparison between standard frame-based and event cameras is shown in Fig. E.1a. Since each pixel is independent and can choose its own operating point, event cameras achieve a very high intra-scene dynamic range (more than 140 dB). However, due to their fundamentally different output (an event stream rather then a sequence of frames), standard computer-vision algorithms cannot be applied to such data directly and new methods to deal with this different way of encoding visual information (temporal contrast rather than absolute brightness) should be devised. As event cameras have become commercially available only in the last few years, *e.g.*the DVS [77] and the DAVIS [19], research on event-based vision is a relatively new topic. The most recent sensor, the DAVIS, has a resolution of 240×180 pixels and, in addition to the events, it also outputs standard grayscale images from the same physical pixels (that we only use for visualization purposes in this work).

Certain applications of event cameras, such as image reconstruction [32, 66, 125] or video decompression [20], require processing all events. However, many applications like visual odometry or object tracking are known from standard cameras to work reliably on corners alone. Corners are useful features as they are well localized, highly informative, and do not suffer from the aperture problem. Additionally, they reduce an image (composed of millions of pixels) to a few hundred measurements. Similarly, we aim at reducing the event stream to a highly-informative *corner event* stream.

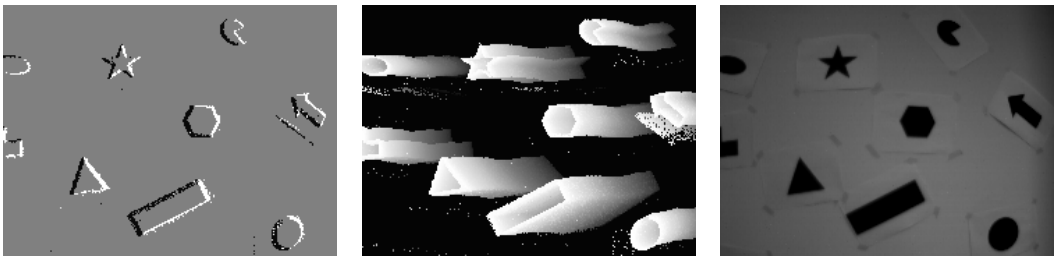
The first method for event-based corner detection was presented by [29]. They estimate the optical flow by fitting planes to the Surface of Active Events (a map of the timestamp of the latest event for each pixel) and searching for intersections. However, as plane fitting is a costly operation, the number of events per second that can be processed is rather limited. A more recent work [142] computes Harris corners on artificial frames generated from the events. While this method shows convincing results, it is not computationally efficient due to the underlying data structure and the required convolutions and matrix multiplications.

Other works focus on feature *tracking* in the event stream. However, they assume to know the shape of the features a priori [72], or they extract features from the frame and only track them using the events [138]. In [151], a probabilistic feature tracking algorithm using Expectation-Maximization is presented. Similarly to [142], they detect Harris corners in artificial frames, but instead of using a binarized frame, they use the event density over a temporal window.

Recently, several methods for camera tracking and visual odometry for event cameras have been presented. In [67] a visual-odometry method was proposed that works in real time, but requires a GPU. Instead of processing single events, EVO [124] accumulates events to build artificial frames as an intermediate step. These works show that real-time performance either requires massive parallel processing power (*e.g.*, a GPU) or departing from the event-based processing paradigm (*i.e.*, each event can change the state of the system). In [70], a visual-odometry algorithm using feature tracks was presented. They showed that feature-based methods work efficiently on event cameras. However, they required frames to detect corners and extract features before they could be tracked with the events.

In this paper, we present a fast method for corner detection in an event stream. Our detector is very efficient due to its design principle, which consists of working on the Surface of Active Events using only comparison operations, as opposed to plane fitting or computing gradients by convolution, as previous works. Our method asynchronously processes event by event with very low latency, thus preserving the characteristics of the event stream. It can serve as a lightweight, low-latency preprocessing step for subsequent higher-level algorithms such as visual odometry, object tracking, or recognition. Our implementation can process more than a million events per second on a single core, and typically reduces the event rate by a factor of 10 to 20.

The remainder of this paper is structured as follows. Section E.2 describes our method, which we evaluate and compare to previous work in Section E.3. Results are discussed and conclusions are drawn in Sections E.4 and E.5, respectively.



(a) Events over the last 33 ms. Positive events (white) and negative events (black)

(b) Surface of Active Events (SAE). Brightness represents time, from past (dark) to present (bright)

(c) Intensity Image (absolute brightness $I(x, y)$ in (E.1))

Figure E.2: Signal used for corner detection: the Surface of Active Events (SAE).

E.2 Method

Inspired by the FAST [126] corner detector for frames, we propose a corner detector for event streams that only uses pixel-wise comparisons. FAST considers a pixel as a corner if n contiguous pixels along a circle around the pixel of interest have all darker (or all brighter) intensity than the center pixel plus a threshold (typically, $n = 9$ on a circle of radius 3 with 16 pixels). In event cameras, brightness is encoded in the form of temporal contrast. More precisely, an event $e = (x, y, t, pol)$ is triggered at a pixel (x, y) at time t if the (logarithmic) brightness I reaches a predefined contrast threshold C (typically 15 %),

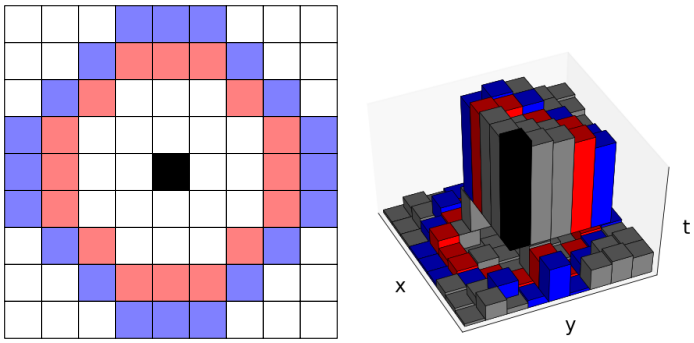
$$I(x, y, t) - I(x, y, t - \Delta t) = pol \cdot C, \tag{E.1}$$

where $t - \Delta t$ is the time when the last event at that pixel was triggered, and pol, the polarity of the event, is the sign of the brightness change. Since visual information is represented by time and there is no notion of frames for event cameras, we propose to operate on the Surface of Active Events (SAE) [9], which is the function given by the timestamp of the most recent event at each pixel:

$$SAE : (x, y) \mapsto t. \tag{E.2}$$

Figure E.2 shows a temporal window of events, the Surface of Active Events, as well as an intensity image for the same moment in time. Similarly to [142], we separate the events by polarity and process them independently.

Since this continuously and asynchronously updated representation is fundamentally different from intensity images, several changes are needed to make a FAST-like detector for event cameras. First, we do not need to iterate over all pixels, but only check the current event using its local neighborhood. This check is performed asynchronously at



(a) Circles of radius 3 and 4 pixels

(b) Visualization of Surface of Active Events (SAE)

Figure E.3: Proposed Method. We compare the timestamps of the latest event of the pixels on two circles (red and blue) around the current event (in black). (a): The inner (red) and outer (blue) circle around the current event (black). (b): Visualization of the Surface of Active Events (SAE) around the current event (black) and of the circles used for the timestamp comparison. In this example, the event under consideration (center pixel) is classified as corner.

the moment the event arrives. Second, the pixel values represent timestamps instead of intensity values and since the current event is considered the center pixel of the local neighborhood it always has the highest timestamp on the SAE. Therefore, comparisons of the pixels on the circle to the center one (as in FAST) are non-informative, and a different spatial comparison pattern (between pixels on the circle only) is required.

Our method analyzes the distribution of timestamps around the current event to decide on the presence of a corner. A moving corner will create, locally, a pixel map of timestamps such as that in Fig. E.3b, with two clearly separated regions (recent vs. old events, i.e., high vs. low values). Hence, we detect corners by searching for contiguous pixels with higher timestamps than the rest. We use circular segments for isotropic response and for efficiency (checking fewer pixels than the whole neighborhood). In contrast to existing methods [142], we completely avoid the computation of derivatives, which are expensive and amplify noise.

More specifically, we define a patch (local spatial neighborhood) of the SAE around the current event. In this patch, we focus on the pixels on two centered, concentric circles of radius 3 and 4 as shown in Fig. E.3. Along each circle, the algorithm searches for segments of contiguous pixels (arcs) that are higher (*i.e.*, they have a more recent timestamp) than all other pixels on the circle. For the inner circle (red), we search for a segment of length between 3 and 6 (pixels). For the outer circle (blue), we search for a segment of length between 4 and 8. If we find such segments on both circles, we consider the current event to be a corner event. In the example in Fig. E.3b, the inner circle (red) and the outer circle have 5 and 6 contiguous pixels that are all higher than the other pixels along the circle, respectively. Therefore, the event in the center pixel is considered a corner.

Experimentally, we found that using additional inner circles (of radius 1 or 2) does not improve detection quality. We suspect that sensor noise is the main issue why corners in current event cameras cannot be located more precisely (feature-track methods [138, 151] also report localization errors in the range of 2 pixels). However, we also found that only using the inner circle of radius 3 provides significantly worse quality compared to using both circles. As can be seen from Fig. E.3a, circles of radius 3 and 4 constitute less than half of the pixels ($36/81 \approx 44\%$) in the 9×9 pixels patch. However, it is this region that we experimentally found to provide the most relevant information for corner detection and localization: larger circles do not provide good localization and smaller circles are not reliable to detect corners as they are more sensitive to sensor noise.

E.3 Evaluation

To evaluate the performance of our method, we first describe how we compute ground truth. Then we review the current state-of-the-art Harris detector [142] and describe our improvement. Finally, we compare the detection performance and runtime of Harris and our method on the Event-Camera Dataset [103].

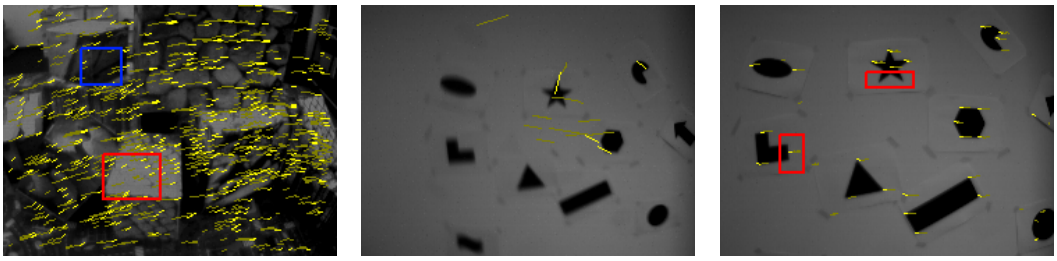
E.3.1 Ground Truth

Ground Truth using Frames

Establishing ground truth using the Lucas-Kanade tracking algorithm [85] on the frames of the DAVIS and interpolating between the frames suffers from severe limitations: (*i*) dynamic range: due to the limited dynamic range of the frames (around 55 dB), no corners are detected and tracked in over- and underexposed areas of the image (see Fig. E.4a), (*ii*) frame rate: due to the limited frame rate of the sensor (around 25 Hz), tracking is lost in high-speed scenarios and linear interpolation is no longer a good approximation (see Fig. E.4b), and (*iii*) corner interpretation: not all elements perceived as corners in the event stream are also recognized as corners in the images, and vice versa, even though they are repeatably detected and well tracked (see Fig. E.4c). We experimented with different corner detectors (Harris and FAST) and different pyramid levels (up to 4 levels). Therefore, we propose a different method for establishing ground truth, which we describe next.

Ground Truth using Feature Tracks

Instead of using frames, we post-process the corner events to find "feature tracks". A feature track is composed of an inner and an outer oblique cylinder in space-time (see Fig. E.5a). We exhaustively search for feature tracks by creating hypotheses using



(a) Dynamic Range **(b)** Frame Rate **(c)** Corner Interpretation

Figure E.4: Issues with image-based ground truth. Frames from the DAVIS with superimposed detected corners (yellow traces) from frame-based Lukas-Kanade corner tracking implementation. (a): Corners in areas with overexposure (red) and underexposure (blue) are not detected in the image (frame from the `boxes` dataset). (b): Too much motion between two frames and motion blur causes Lucas-Kanade tracking to fail (frame from the `shapes` dataset). (c): Not the same corners are detected using the frames and the events (highlighted in red) (frame from the `shapes` dataset).

two corner events and checking whether there are enough corner events in the inner cylinder and few corner events in the outer cylinder. We used 3 and 5 pixels for the inner and outer radius, respectively, a minimum of 30 inner events, and maximum ratio of outer-to-inner events of 25 %. We then consider all corner events belonging to such a feature track as inliers of the hypothesis and label them as correct corners.

E.3.2 Event-based Harris Detector

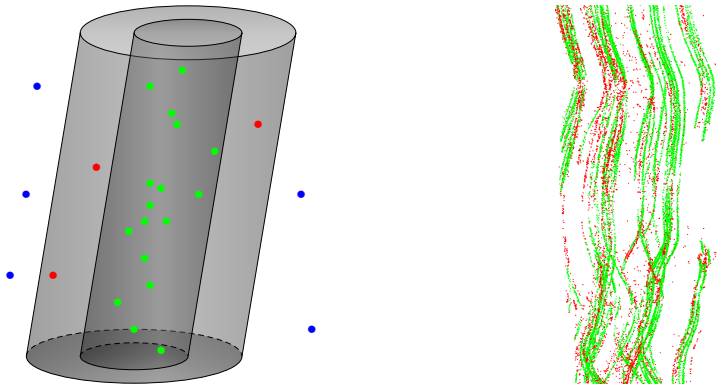
We compare our method with the event-based adaptation [142] of the Harris detector, which we describe here for completeness. Their method binarizes the SAE by the newest N events (depending on the experiment, they choose $N = 1000$ or $N = 2000$). Let Σ_b be a binary patch centered around the latest event, where 0 and 1 indicate the absence and presence of an event, respectively. Compute $I_x = \Sigma_b * G_x$ and $I_y = \Sigma_b * G_y$ as convolutions of the patch with 5×5 Sobel kernels G_x and $G_y = G_x^\top$, respectively. Compute Harris matrix

$$M = \sum_{e \in \Sigma_b} g(e)\ \nabla I(e) \nabla I^\top(e), \tag{E.3}$$

where g is a Gaussian weighting function with spread $\sigma = 1$ pixel, $\nabla I(e) = (I_x(e), I_y(e))^\top$ is the gradient at pixel e, and the 2×2 matrix $\nabla I(e) \nabla I^\top(e)$ is the point-wise structure tensor. Finally, the Harris score is computed as

$$H = \det(M) - k \cdot \mathrm{trace}(M)^2, \tag{E.4}$$

where $k = 0.04$ is a user-defined parameter. The event at the center of the patch is classified as a corner if its score H is large than a threshold S.



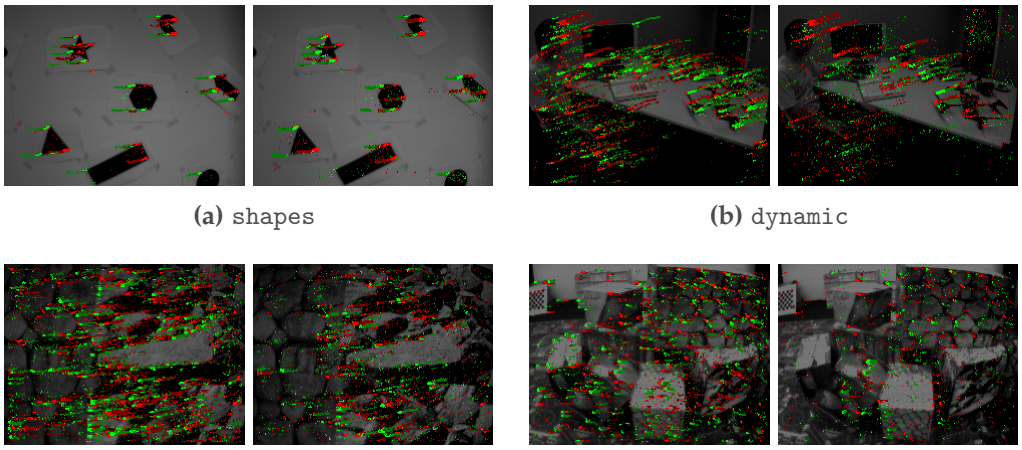
(a) Feature Track **(b)** Labeled Data

Figure E.5: Ground truth using feature tracks. (a) Feature tracks shown in space-time (time going upwards) represented by an inner and outer oblique cylinder. Points represent events that have been detected as corner events. Only if few corner events fall within the outer cylinder (red), the corners in inner cylinder are considered as correct detections (green). (b) Visualization of labeled data where each dot represents a corner detection. Green dots are part of a feature track (and therefore labeled as "true" corner), whereas red corners are considered false detections.

Spatially-Adaptive Harris Method. We propose the following improvement to the above-mentioned event-based Harris detector. Instead of choosing the newest N events for the whole image plane, which depends on the amount of texture in the scene, we choose the number of newest events locally, N_l, around the event under consideration. This choice is more sensible since only the latest events around the current one are relevant for deciding whether that event is a corner or not. Hence, our modified Harris detector adapts to the local visual information and is independent of the scene and the sensor size. We found that a patch of 9×9 pixels, the latest $N_l = 25$ events therein, and a threshold of $S = 8$ gave the best performance over a wide variety of datasets. Note that this is the same patch size as the one used in our proposed FAST-inspired corner-detection method (see Section E.2).

E.3.3 Detector Performance

We compare the performance of our method with the spatially-adaptive Harris method described above. We evaluate the detectors on a representative subset of the datasets provided by the publicly available Event-Camera Dataset [103]. Each dataset is approximately one minute long and contains tens of millions of events. The scenes in the dataset range from simple shapes to natural and office environments. The motion speed, and therefore also the event rate, steadily increases during the datasets, reaching top values of over $3\,\mathrm{m/s}$ and $900\,^\circ/\mathrm{s}$, corresponding to activity peaks of 8 million



(a) `shapes` **(b)** `dynamic`

(c) `poster` **(d)** `boxes`

Figure E.6: Visualization of corner detections of the last 33 ms and 100 ms in bright and dark color, respectively. Left and right images show the detections of Harris and our method, respectively. The datasets are from the Event-Camera Dataset [103]. The images are only shown for visualization and not used in either method. Color indicates the polarity of the corner events: green and red are events with positive and negative polarity, respectively.

events per second.[1]

The results are summarized in Table E.1 and report the reduction rate in percentage (Red.) and the percentage of corner detections that could be matched to a feature track (FT, cf. Section E.3.1). Figure E.6 shows snapshots from both methods for all scenes overlaid on the frame. Figure E.1b shows the corners in space-time together with the event stream for the `shapes` dataset during an interval of 1 s. Our method performs slightly worse than spatially-adaptive Harris on almost all datasets, but runs more than an order of magnitude faster, as shown in the next section (see Table E.2). Both methods show the same trend: on scenes with low texture (such as `shapes` that contains only black-and-white patterns or `dynamic` that contains a desk, screen, books, and a moving person), both methods perform very well. On more finely-textured scenes (such as `boxes` and `poster` that contain fine-grained natural pattern), fewer feature tracks can be found in the corner event stream. This does not necessarily mean that the detections are wrong, but rather that there might be corner-like structures very close in the image that cannot be separated well with the proposed ground-truth labeling technique. Further, the detector performance does not significantly depend on the motion type (rotation, translation, or 6-DOF), but rather on the level of texture in the scene.

[1] Sampled at 1 ms intervals.

Texture	Dataset	Harris [142] Red.	Harris [142] FT	Ours Red.	Ours FT
low	shapes_rotation	92.7	74.1	88.9	74.7
	shapes_translation	91.7	78.3	87.8	77.5
	shapes_6dof	90.6	77.1	87.0	76.8
medium	dynamic_rotation	95.1	53.3	96.4	46.4
	dynamic_translation	95.3	62.1	96.7	52.1
	dynamic_6dof	95.4	55.9	96.4	49.4
high	poster_rotation	92.6	35.3	95.7	30.5
	poster_translation	92.3	39.5	95.8	35.9
	poster_6dof	92.4	35.3	95.6	32.1
high	boxes_rotation	92.1	32.9	96.7	25.2
	boxes_translation	92.4	37.0	96.7	30.5
	boxes_6dof	92.7	34.4	96.8	26.7

Table E.1: Performance of Harris and our detector expressed as reduction rate (Red.) of the event stream and matched Feature Tracks (FT). Values are given in percentages.

E.3.4 Computational Performance

A major advantage of our algorithm is its runtime. Due to the asynchronous nature of event cameras, the event rate depends on the scene, the motion, and the sensor parameters (biases). The event rate in the Event Camera Dataset [103] is in the range of a few million of events per second (peaks of 8 million events per second). Our algorithm runs at 780 ns per event, allowing rates of up to 1.2 million events per second—more than an order of magnitude higher than previous methods. The results are summarized in Table E.2. We used a single core of an Intel i7-3720QM CPU at 2.60 GHz for all experiments.

Method	Time per event [µs]	Max. event rate [e/s]
Harris [142]	11.6	86,230
Ours	0.78	1,275,000

Table E.2: Runtime comparison per event and corresponding maximum event rate.

Since our method uses only a small local neighborhood of the events, parallelization is straightforward, if needed, with very little overhead. While this argument applies also to the Harris detector, the runtime per event remains critical to achieve *low-latency* performance: the DAVIS has a latency of 3 µs [19]. While the application of Harris triples this latency (11.6 µs), our method only yields around 30 % additional latency to the overall system (0.78 µs). The Harris method is slower for two main reasons: (*i*) a sort operation is required to find the latest N events on the SAE[2]; (*ii*)

[2]Just keeping the last N events in a queue is not equivalent, because pixels often fire more than one

the computation involves convolutions (Sobel operator to compute ∇I) and matrix multiplications (Gaussian weighting). Our method, instead, works only by direct, pixel-wise comparisons on the SAE. Thus, there are no expensive floating-point or sorting operations carried out on the pixel values. Additionally, as mentioned in Section E.2 (Fig. E.3a), our method is also fast because it processes only the most relevant part of the patch for the current event, which accounts to less than half of the pixels in the patch.

E.4 Discussion

The early reduction of the event stream to a *corner event* stream has several advantages. First, the corner detector acts as a filter: letting through only the most informative (i.e., less ambiguous) and well-localized events, and reducing, by more than an order of magnitude, the amount of data that must be processed at later stages in the pipeline, at little computational cost. Second, the low-latency and asynchronous nature of event camera output is maintained because each event is processed as soon as it is received. Since our algorithm runs very fast, very little additional latency is introduced. Third, the event-based paradigm of processing data on an event-by-event basis is preserved since we decide whether an event is a corner immediately and only using past events in a local neighborhood.

While the corner detection quality is slightly worse than an improved version of a state-of-the-art method, its computational performance is more than an order of magnitude faster. However, since both the Harris detector and our method operate on the same signal (the SAE), it would also be feasible to refine our corner event detections by post-processing them with the event-based Harris method.

E.5 Conclusion

We present a fast corner-detection algorithm that works on the asynchronous output stream of event cameras and preserves its low-latency and asynchronous characteristics. Our method reduces the event rate by 90%-95% and achieves a number of correctly-tracked features close to a state-of-the-art event-based corner detector (less than 10% difference). Since our method works directly on the Surface of Active Events using only binary comparisons, the processing time per event is very little and millions of events can be processed per second on a single core, which is more than 10 times faster than state-of-the-art methods. If needed, our method can be parallelized with almost no overhead since it uses only local information. Furthermore, as the resolution of future event cameras steadily increases, the event rate will also increase, and our algorithm will become more relevant to convert the event stream into a more manageable stream

event. For Harris to work best, we need the last N events from *distinct* pixels.

of informative and well-localized events.

Future work will include improving the detection quality further and investigation of non-maximum suppression methods, which is non-trivial due to the asynchronous nature of the events.

Acknowledgements. We thank Guillermo Gallego, Arren Glover and Valentina Vasco for valuable discussions.

F Event-based Feature Tracking

D. Tedaldi, G. Gallego, E. Mueggler, and D. Scaramuzza. "Feature Detection and Tracking with the Dynamic and Active-pixel Vision Sensor (DAVIS)". in: *Int. Conf. Event-Based Control, Comm. Signal Proc. (EBCCSP)*. Krakow, Poland, June 2016, pp. 1–7. DOI: 10.1109/EBCCSP.2016.7605086

Feature Detection and Tracking with the Dynamic and Active-pixel Vision Sensor (DAVIS)

David Tedaldi, Guillermo Gallego, Elias Mueggler and Davide Scaramuzza

Abstract — Because standard cameras sample the scene at constant time intervals, they do not provide any information in the blind time between subsequent frames. However, for many high-speed robotic and vision applications, it is crucial to provide high-frequency measurement updates also during this blind time. This can be achieved using a novel vision sensor, called DAVIS, which combines a standard camera and an asynchronous event-based sensor in the same pixel array. The DAVIS encodes the visual content between two subsequent frames by an asynchronous stream of events that convey pixel-level brightness changes at microsecond resolution. We present the first algorithm to detect and track visual features using both the frames and the event data provided by the DAVIS. Features are first detected in the grayscale frames and then tracked asynchronously in the blind time between frames using the stream of events. To best take into account the hybrid characteristics of the DAVIS, features are built based on large, spatial contrast variations (i.e., visual edges), which are the source of most of the events generated by the sensor. An event-based algorithm is further presented to track the features using an iterative, geometric registration approach. The performance of the proposed method is evaluated on real data acquired by the DAVIS.

F.1 Introduction

Feature detection and tracking are the building blocks of many robotic and vision applications, such as tracking, structure from motion, place recognition, etc. Extensive research has been devoted to feature detection and tracking with conventional cameras, whose operation principle is to temporally sample the scene at constant time intervals. However, conventional cameras still suffer from several technological limitations that prevent their use in high speed robotic and vision applications, such as autonomous cars and drones: (*i*) low temporal discretization (i.e., they provide no information during the blind time between consecutive frames), (*ii*) high redundancy (i.e., they wastefully transfer large amounts of redundant information even when the visual content of the scene does not change), (*iii*) high latency (i.e., the time needed to capture and process the last frame). Since the agility of an autonomous agent is determined by the latency and temporal discretization of its sensing pipeline, all these advantages put a hard bound on the maximum achievable agility of a robotic platform.

Bio-inspired event-based sensors, such as the Dynamic Vision Sensor (DVS) [78, 77, 36] or the Asynchronous Time-based Image Sensor (ATIS) [118, 117, 116], overcome the above-mentioned limitations of conventional cameras. In an event-based sensor, each pixel operates independently of all other pixels, and transmits asynchronously pixel-level brightness changes, called "events", at microsecond resolution at the time they occur. Hence, an event camera virtually eliminates latency and temporal discretization. Also, it avoids redundancy, as no information is transmitted if the scene does not change. However, this comes at a price: the output of an event camera (a stream of events) is fundamentally different from that of conventional cameras; hence, mature computer vision algorithms cannot be simply adapted, and new, event-driven algorithms must be developed to exploit the full potential of this novel sensor.

More recently, hybrid vision sensors that combine the benefits of conventional and event-based cameras have been developed, such as the Dynamic and Active-pixel VIsion Sensor (DAVIS) [19]. The DAVIS implements a standard grayscale camera and an event-based sensor in the same pixel array. Hence, the output consists of a stream of asynchronous high-rate (up to 1 MHz) events together with a stream of synchronous grayscale frames acquired at a low rate (on demand and up to 24 Hz).

We present the first algorithm to detect features from the DAVIS frames and perform event-driven high-temporal resolution tracking of these features in the blind time between two frames. The key challenge consists of designing an algorithm that best takes into account the hybrid characteristics of the DAVIS output to solve the detection-and-tracking problem (Fig. F.1). Since events are generated by changes of brightness in the scenes, features are built based on large, spatial contrast variations (i.e., visual edges), which are the source of most of the events generated by the sensor. An event-based algorithm is further presented to track the features using an iterative, geometric

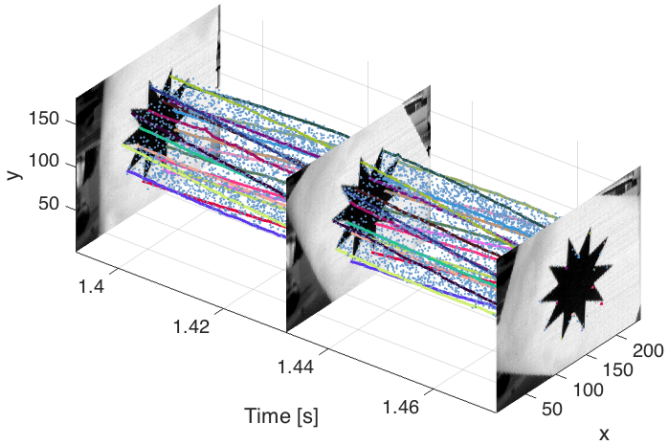


Figure F.1: Spatio-temporal view of the output of the DAVIS (frames and events) and the trajectories of the tracked features (in different colors, one for each feature). In this example, the scene consists of a rotating object. The motion in the blind time between consecutive frames is accurately tracked using the stream of events; e.g., rotation is clearly visible in the spiral-like trajectories of the event-based tracked features. To facilitate the visualization, only 10% of the events is displayed.

registration approach.

The paper is organized as follows. Section F.2 reviews the related work on event-based feature detection and tracking. Section F.3 describes the DAVIS sensor. Section F.4 describes the proposed detection and tracking algorithm. Section F.5 presents the experimental results. Finally, section F.6 draws the conclusion and gives future perspectives.

F.2 Related Work

F.2.1 From Frame-based to Event-based Tracking

Feature detection and tracking methods for frame-based cameras are well-known [85, 30, 134]. The pixel intensities around a corner point are used as a template that is compared frame-by-frame with the pixels around the estimated position of the corner point. The photometric error is then used to update the parameters describing the position and warping of the template in the current frame. These appearance-based methods do not apply to event cameras; however, the approach of using a parametric template model and updating its parameters according to data fitting still applies.

From a high-level point of view, two relevant questions regarding event-based tracking are *what to track* and *how to track*. The first question refers to how are the objects of interest modeled in terms of events so that object instances can be detected in the event stream. The answer to this question is application dependent; the object of interest is

usually represented by a succinct parametric model in terms of shape primitives. The second question, "how to track?", then refers to how to update the parameters of the model upon the arrival of data events (caused by relative motion or by noise). For a system that answers the aforementioned questions, a third relevant question is "*what kind of object motions or distortions can be tracked?*" The above-mentioned three questions are key to understand existing tracking approaches.

F.2.2 Event-based Tracking Literature

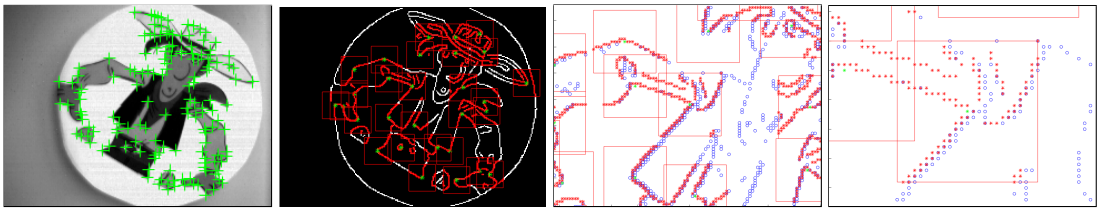
Early event-based feature trackers were very simple and focused on demonstrating the low-latency and low-processing requirements of event-driven vision systems, hence they tracked moving objects as clustered blob-like sources of events [80, 79, 35, 34, 113] or lines [31].

Accurate tracking of general shapes can be performed by continuously estimating the warping between the model and the events. This has been addressed and demonstrated for arbitrary user-defined shapes using event-based adaptions of the Iterative Closest Point (ICP) algorithm [108], gradient descent [109], or Monte-Carlo methods [72] (i.e., by matching events against a uniformly-sampled collection of rotated and scaled versions of the template). Detection and tracking of locally-invariant features, such as corners, directly from event streams has been addressed instead in [29].

Notice, however, that all above-mentioned papers were developed for event-only vision sensors. In this paper, we build upon these previous works and present the first algorithm to automatically detect features from the DAVIS frames and perform event-driven high-temporal resolution tracking of these features in the blind time between two frames.

F.3 The Dynamic and Active-pixel VIsion Sensor

The DAVIS [19] is a novel vision sensor combining a conventional frame-based camera (active pixel sensor - APS) and a DVS in the same array of pixels. The global-shutter frames provide absolute illumination on demand and up to 24 Hz, whereas the event sensor responds asynchronously to pixel-level brightness changes, independently for each pixel. More specifically, if $I(t)$ is the illumination sensed at pixel (x, y) of the DVS, an event is triggered if relative brightness change exceeds a global threshold: $|\Delta \ln I| := |\ln I(t) - \ln I(t - \Delta t)| > C$, where Δt is the time since the last event was triggered (at the same pixel). An event is a tuple $e = (x, y, t, p)$ that conveys the spatio-temporal coordinates (x, y, t) and sign (i.e., polarity $p = \pm 1$) of the brightness change. Events are time-stamped with microsecond resolution and transmitted asynchronously when they occur, with very low latency 15 µs. The DAVIS has a very high dynamic



(a) (b) (c) (d)

Figure F.2: Feature detection and tracking. (a) Frame with centers of detected features (green crosses). (b) Edge map (black and white) and square patches defining the features (i.e, model point sets, in red) (b)-(c) Zoomed views of the data point sets (i.e., events; blue circles) and model point sets (red stars) of the features, shortly after initialization.

range (130 dB) compared with the 70 dB of high-quality, traditional image sensors. The low latency, the high temporal resolution, and the very high dynamic range make the DAVIS extremely advantageous for future robotic applications in uncontrolled natural lighting, i.e., real-world scenarios.

A sample output of the DAVIS is shown in Fig. F.1. The spatial resolution of the DAVIS is 240×180 pixels. This is still limited compared to the spatial resolution of state-of-the-art conventional cameras. Newer sensors, such as the color DAVIS (C-DAVIS) [75] will have higher spatial resolution (640×480 pixels), thus overcoming current limitations.

F.4 Feature Detection and Tracking with the DAVIS

Since events are generated by changes of brightness, this implies that only edges are informative. Intersecting edges create corners, which are "features" that do not suffer from the aperture problem and that have been proven to be optimally trackable in frame-based approaches [134]. Therefore, event-based cameras also allow for the perception of corners, as shown in [29]. We exploit these observations to extract and describe features using the DAVIS frames, and then track them using the event stream, as illustrated in Fig. F.2. Our method builds upon the customized shapes in [72] and the update scheme in [108]. The technique comprises to main steps: feature detection and tracking, as we detail in the next sections.

F.4.1 Feature Detection From Frames

The absolute brightness frames of the DAVIS are used to detect edges (e.g., Canny's method [23]) and corners (e.g., Harris detector [58]). Around the strongest corners, we use the neighboring pixels of the Canny edge-map to define patches containing the dominant source of events. We simplify the detection by converting the edge-map patches to binary masks indicating the presence (1) or absence (0) of an edge. The

Algorithm 1 High temporal resolution tracking

Feature detection:
- Detect corner points on the frame (Harris detector).
- Run Canny edge detector (returns a binary image, 1 if edge pixel; 0 otherwise).
- Extract local edge-map patches around corner points, and convert them into *model* point sets.

Feature tracking:
- Initialize a *data* point set per patch

for each incoming event **do**
  - Update the corresponding data point set.
  for each corresponding data point set **do**
    - Estimate the registration parameters between the data and the model point sets.
    - Update registration parameters of the model points.

binary masks define the interest shapes for tracking in terms of 2D point sets, called "*model* point sets". These steps are summarized at the beginning of Algorithm 1.

We use square patches of the same size, which is an adjustable parameter. However, it is straightforward to extend the method to consider different aspect ratios and sizes.

Frames are not required to be provided at a constant rate since they are only used to initialize features; they can be acquired on demand to replace features that are lost or fall out of the field of view of the sensor.

F.4.2 Feature Tracking From the Event Stream

Extracted features are tracked using subsequent events from the DAVIS. The input to the event-based tracking algorithm consists of multiple, local model point sets. The second part of Algorithm 1 summarizes our tracking strategy.

Sets of Events used for Feature Registration

For every feature, we define a *data* point set of the same size as the *model* point set. Therefore, the size can be different for every feature, depending on edge information. Data point sets consist of local space-time subsets of the incoming events: an event is inserted in a data point set if the event coordinates are inside the corresponding patch. Once a data point set has been filled, registration of the point sets can be done. Hence, a data point set defines the set of events that are relevant for the registration of the associated feature. Registration is carried out by minimization of the distance between the data and the model point sets, as explained next. Data point sets are continuously updated: the newest event replaces the oldest one and the registration

iteration proceeds.

This procedure is event-based, i.e., the parameters of the tracked feature are updated every time an incoming event is considered relevant for that feature. The algorithm is asynchronous by design, and can process multiple features simultaneously. Several strategies to assign an incoming event to one or more overlapping patches can be used, in a way similar to [109]. We updated all models around the ambiguous event.

Registration

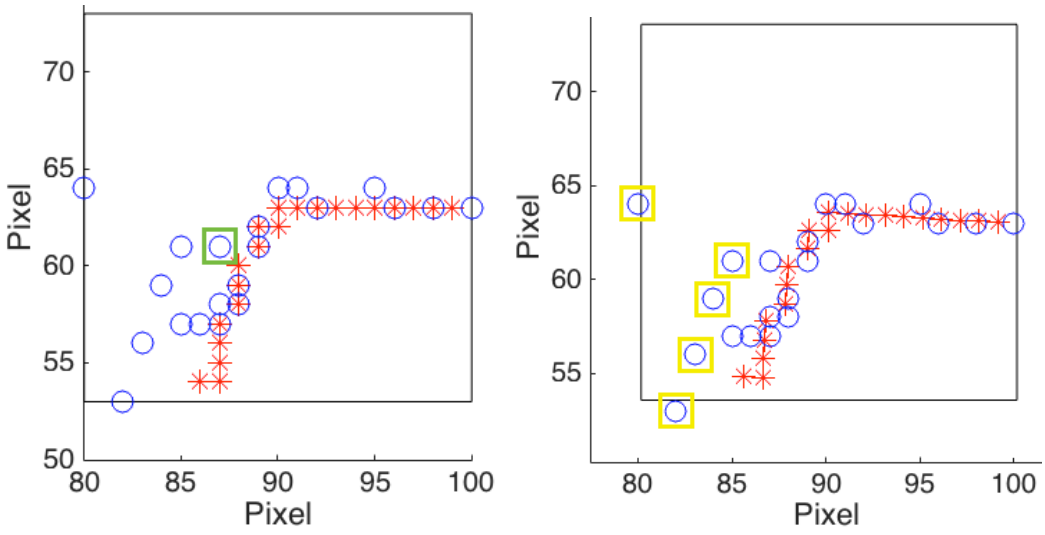
The data point set from the events, $\{\mathbf{p}_i\}$, is registered to the model point set (feature), $\{\mathbf{m}_i\}$, by minimization of the Euclidean distance between the sets, and including outlier rejection:

$$\arg\min_{\mathtt{A}} \sum_{(\mathbf{p}_i,\mathbf{m}_i)\in \text{Matches}} \|\mathtt{A}(\mathbf{p}_i) - \mathbf{m}_i\|^2, \tag{F.1}$$

where $\mathtt{A}$ is the registration transformation between the matched point sets. For simplicity, we choose $\mathtt{A}$ within the class of Euclidean motions, but the method can be extended to more complex transformations. We choose the iterative closest point algorithm (ICP) [12] to minimize (F.1). Matches $\mathbf{p}_i \leftrightarrow \mathbf{m}_i$ are established according to nearest neighbor; a predefined distance of 2 pixels between the events in the data point set and the model point set is used for outlier rejection. Each algorithm iteration has three stages: first, candidate matches are established, then the geometric transformation is estimated, and, finally, the transformation is applied to the model point set. The operation proceeds until the error difference between two consecutive iterations is below a certain threshold.

Fig. F.3a shows both the model and the data point sets. When a new event arrives, the geometric transformation that defines the tracker is updated according to the minimization of (F.1). The result is depicted in Fig. F.3b. By discounting the points classified as outliers by the algorithm (in yellow), registration is accurate. Feature trajectories are given by the positions of the features returned by the registration step.

Due to the high temporal resolution of the DAVIS, the transformation between consecutive events (in the same feature) is close to the identity (Fig. F.3b), and so, our method yields good results even after a single iteration. In practice, it is more efficient to compute the registration transformation every M events, e.g., of half the size of the model point set.



(a) Before registration **(b)** After registration

Figure F.3: A feature tracker, with the model point set (in red), the data point set (in blue). Same color notation as in Figs. F.2c-F.2d. The black square represents the patch around the model point set. (a) Before registration: the current event (in green) updates the data point set and is used for registration of the point sets. (b) After registration: the events marked in yellow are classified as outliers, and the registration parameters are updated, aligning the model and data point sets.

F.5 Experiments

We present the tests performed to validate the algorithm and to study its performance in different scenes with increasing level of complexity: a very large contrast (i.e., black and white) scene, a piecewise constant scene (a cartoon), and a natural scene (the leaves of a tree; see Fig. F.11). The first scene depicts a black star on a white background; this scene has sharp transitions between the two intensity levels, showing clear edges (Fig F.1) and well-localized features. The second scene consists of a cartoon image with piecewise constant regions (Fig F.8a); intensity is concentrated in a few grayscale levels and there are moderately abrupt transitions between them. The third scene is a representative of a natural image, rich in texture and brightness changes of different magnitudes (Fig. F.11a) coming from the leaves of a tree. The scene datasets show dominant translational and rotational motions.

We used patches of 25×25 pixels, which is approximately $1/10$ of the image width. This size was empirically found to be best for a broad class of scenes.

We measured the tracking error over time. The tracking error is computed against ground truth, which was generated using a frame-based Lucas-Kanade tracker [85] and linearly interpolating the feature motion in the time interval between frames. The ground truth has sub-pixel accuracy. Features were detected in the first frame ($t = 0$)

and then tracked over the entire sequence using only events. In all scenes, the mean tracking error is less than 2 pixels. Notice that in spite of the sequences being short, they contain several million events.

F.5.1 Large-Contrast Scene ("Star")

Translation

We moved a 10-point star sideways, back and forth, in front of the DAVIS. The algorithm detected one feature every two edges of the star (so there are 20 corners). The mean tracking error plot for all features is shown in Fig. F.4. As it can be observed in the plot, there is a short pause after 1.5 s, marked with a constant error, before changing direction. In this interval, there are virtually no events, and so, the feature tracks do not move, waiting to observe new events in order to keep updating the features' position and orientation.

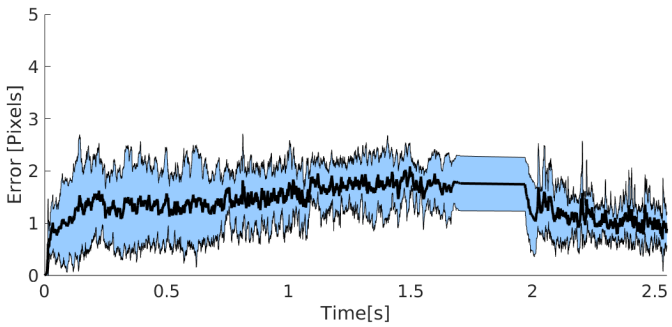


Figure F.4: *Star* (translation) dataset: feature tracking error of our event-based algorithm on translational motion facing the star shown in Fig. F.1. The mean tracking error of all features is marked in black. The blue bands around the mean indicate the ± 1 standard-deviation confidence interval. The overall mean error is 1.52 pixels.

High-Speed Rotation

Next, we made the 10-point star pattern rotate, accelerating from rest to 1.600 °/s (see Fig. F.5) using a electro-mechanical device. Observe that, while the overall motion of the star is a rotation approximately around the center of the image, features are much smaller than the whole star and so they only "see" parts of the peaks, consisting of at most two lines. Nevertheless, these very simple features are able to track the motion of the scene very well.

Because of the offset of the features from the rotation center of about 65 pixels, the features translate at high speeds (more than 1,800 pixels/s on the image plane). For this dataset, ground truth was annotated since the frame-based solution failed: since the star is rotating by up to two points (peaks) between frames, aliasing prevented from

obtaining the correct motion. This speed at which there is frame-based aliasing is not a problem for event-based tracking due to the microsecond temporal resolution of the pixels. The orientation error of the features is shown in Fig. F.6. The mean orientation error remains below 20° over more than two full revolutions, leading to a relative error of 2.3 %. The feature tracks form spirals in image space-time, as shown in Fig. F.7. All of the 20 features (one per vertex) of the 10-point star were accurately tracked during the entire sequence.

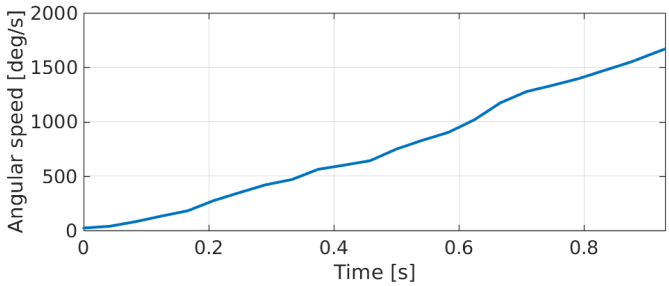


Figure F.5: *Star* (rotation) dataset: angular speed of rotating star. With an approximately constant acceleration (i.e., linear velocity profile), the angular speed reaches more than 1,600 °/s.

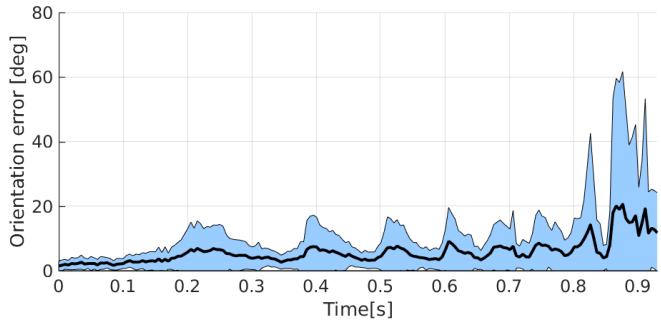


Figure F.6: *Star* (rotation) dataset: feature tracking error of our event-based algorithm on the dataset shown in Fig. F.1. The mean tracking error of all features is marked in black. The blue bands around the mean indicate the ±1 standard-deviation confidence interval. The overall mean error is 6.3°.

F.5.2 Cartoon Scene ("Lucky Luke")

Fig. F.8 shows several snapshots of the tracked features on a sequence of the cartoon scene. The dominant motion is a horizontal translation, back and forth. We observe that 81 features well distributed in the object are correctly tracked throughout the event stream. The tracking error is reported in Fig. F.9. As observed in the plot, there is a short pause after 1 s (constant error), before changing the motion direction. A slight increase of the error can be observed when the motion resumes. However, the mean error in this part of the motion is less than 2 pixel and the overall mean error is small: 1.22 pixel. Tracking in this scene is very good due to two reasons: (*i*) most of the events

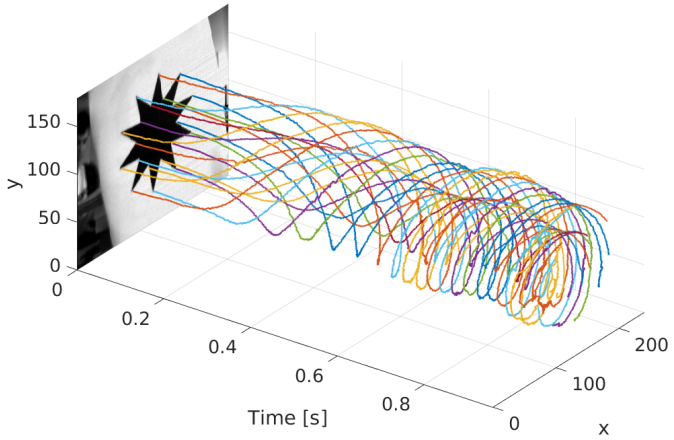
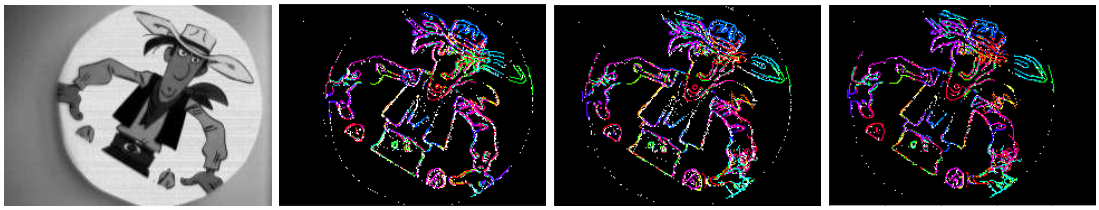


Figure F.7: *Star* (rotation) dataset: space-time locations of the features. Due to the rotation of the star, the feature tracks form spirals. The spiral step gets smaller as the angular speed increases, in this case, with constant acceleration.



(a) DAVIS frame for initialization.

(b) Events (white over black) and features (solid colors) shortly after initialization.

(c) Features during motion.

(d) Features during motion, at a later time than (c).

Figure F.8: *Lucky Luke* (cartoon) dataset. The DAVIS is moving sideways while viewing a natural scene consisting of leaves (a). Individually tracked features (model point sets) are marked in different colors in (b) to (d).

are located at the strong edges, which are captured by the features, and regions of constant intensity do not generate events. (*ii*) there are more than two edges per feature, and with a complex shape (edges in several directions) that make them distinctive for alignment. The tracked features in image space-time are shown in Fig. F.10.

F.5.3 Natural Scene ("Leaves")

In natural scenes (Fig. F.11a), edges can have all sort of different magnitudes, but our features still track the most dominant ones. In this experiment, we moved the DAVIS in front of a natural scene containing both edges (mostly at leave borders) and smooth intensity variations (within the leaves). The motion was oscillatory and predominantly translational (Fig. F.11). Fig. F.12 shows the feature position error; the mean error is

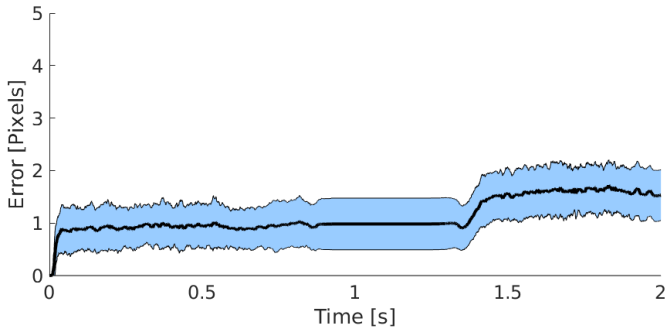


Figure F.9: *Lucky Luke* dataset: feature tracking error of our event-based algorithm. The mean tracking error of all features is marked in black. The blue bands around the mean indicate the ± 1 standard-deviation confidence interval. The overall mean error is 1.22 pixels.

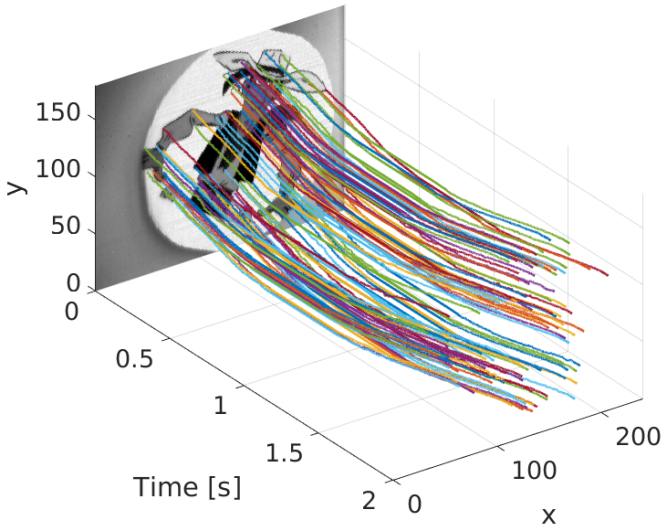
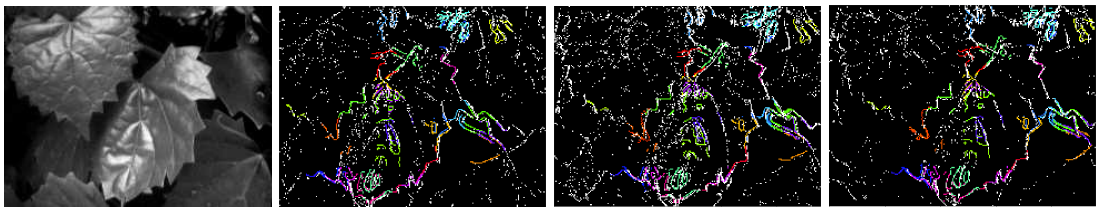


Figure F.10: *Lucky Luke* dataset: space-time view in the image plane of the tracked features' trajectories. The sideways motion is clearly visible in the feature trajectories.

1.48 pixels.

Feature tracks in image space-time are shown in Fig. F.13. Fewer features are tracked compared to the simpler scenes (large contrast and cartoon) because of two reasons: (*i*) the detected features are based on a binary edge-map of the scene (resulted from the Canny detector), but such binary map is an exact representation of the underlying grayscale scene only if the contrast is sufficiently large. (*ii*) we do not model many of the non-linearities of the DAVIS, such as non-white noise and other dynamic properties, which have a larger effect on natural scenes than in simpler ones because events are triggered all over the patches. Notwithstanding, for some robotics applications there is no need to track many features; for example, in perspective-N-point problems) it is sufficient to track as few as three features [68].



(a) DAVIS frame for initialization.

(b) Events (white over black) and features (solid colors) shortly after initialization.

(c) Features during motion.

(d) Features during motion, at a later time than (c).

Figure F.11: *Leaves* dataset. The DAVIS is moving sideways while viewing a natural scene consisting of leaves (a). Individually tracked features (model point sets) are marked in different colors in (b) to (d).

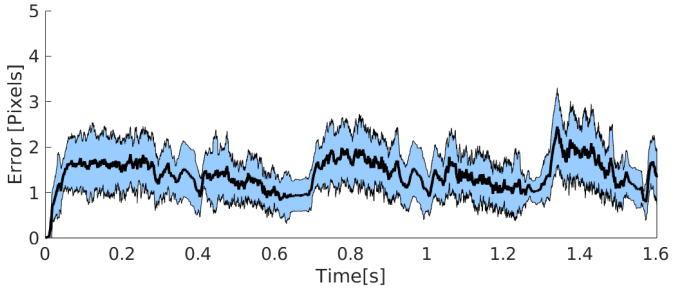


Figure F.12: *Leaves* dataset: feature tracking error of our event-based algorithm. The mean tracking error of all features is marked in black. The blue bands around the mean indicate the ± 1 standard-deviation confidence interval. The overall mean error is 1.48 pixels.

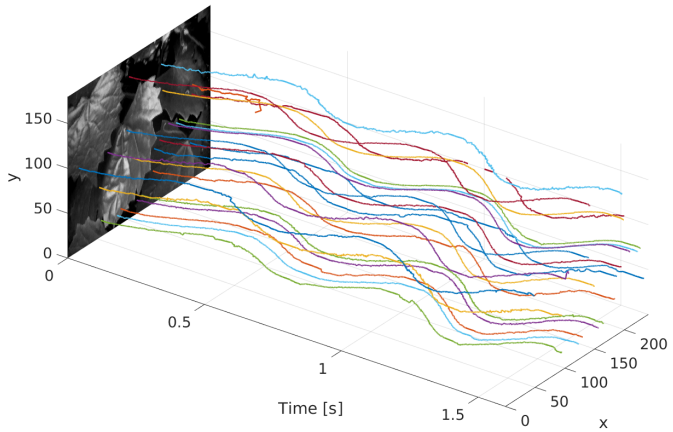


Figure F.13: *Leaves* dataset: space-time view in the image plane of the tracked features' trajectories. The oscillating motion of the DAVIS is correctly captured by the feature trajectories.

Notice that, overall, all the experiments show that our proposed and automatically-detected features can be tracked for a considerable amount of time, much larger than

the time between consecutive frames. Hence, lost features (e.g., falling out of the field of view) could be replaced by new ones that would be initialized using frames at a much lower rate (e.g. 1 Hz) or on demand.

Our method has been tested with real data, with different types of motion, and the results show accurate tracking (less than 2 pixels mean error). Better and more accurate results could be obtained by incorporating the edge strength and the event generation model.

F.6 Conclusions

We have developed a high-temporal tracking algorithm for hybrid sensors such as the DAVIS. We used principled arguments of event data generation to justify our choice of relevant features to track, and proposed a pipeline to extract those features from the frames. Then we used an event-based tracking algorithm that exploits the asynchronous and high temporal resolution of the event stream. In our method, features are automatically and accurately initialized, and are adapted to the scene content, thus overcoming the shortcomings of existing methods. We tested the algorithm on real data from several sequences, and the results validate the approach.

Inspired by the achieved tracking accuracy, we intend to build a visual-odometry pipeline on top of this event-based feature tracking method. Finally, the frames used in our algorithm to initialize the features suffer from motion blur and limited dynamic range, as in any standard camera. To overcome these limitations, we plan to investigate methods to extract features directly from the event stream.

Acknowledgement

We thank Tobi Delbruck for providing the DAVIS240C and Beat Kueng for helping with data recording.

G Sparse Visual Odometry with Feature Tracks

B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza. "Low-latency Visual Odometry using Event-based Feature Tracks". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. Daejeon, Korea, Oct. 2016, pp. 16–23. DOI: 10.1109/IROS.2016.7758089

Low-latency Visual Odometry using Event-based Feature Tracks

Beat Kueng, Elias Mueggler, Guillermo Gallego and Davide Scaramuzza

Abstract — New vision sensors, such as the Dynamic and Active-pixel Vision sensor (DAVIS), incorporate a conventional camera and an event-based sensor in the same pixel array. These sensors have great potential for robotics because they allow us to combine the benefits of conventional cameras with those of event-based sensors: low latency, high temporal resolution, and high dynamic range. However, new algorithms are required to exploit the sensor characteristics and cope with its unconventional output, which consists of a stream of asynchronous brightness changes (called "events") and synchronous grayscale frames. In this paper, we present a low-latency visual odometry algorithm for the DAVIS sensor using event-based feature tracks. Features are first detected in the grayscale frames and then tracked asynchronously using the stream of events. The features are then fed to an event-based visual odometry algorithm that tightly interleaves robust pose optimization and probabilistic mapping. We show that our method successfully tracks the 6-DOF motion of the sensor in natural scenes. This is the first work on event-based visual odometry with the DAVIS sensor using feature tracks.

G.1 Introduction

Vision systems for robotics are currently dominated by methods designed for conventional, frame-based cameras, which acquire entire images of the scene at fixed rates.

Recently, bio-inspired silicon retinas [77, 116] have been developed to overcome some of the limitations of frame-based cameras. These sensors constitute a paradigm shift since

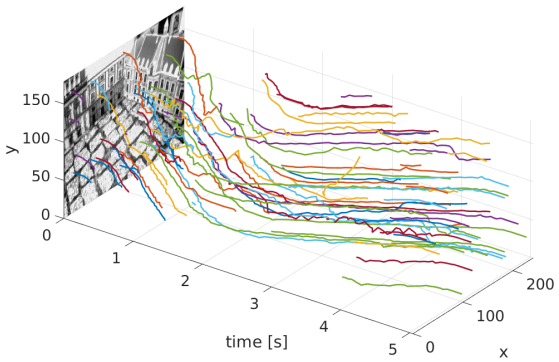


Figure G.1: Space-time view in the image plane of the tracked features' trajectories used for visual odometry. Features are tracked using the events produced by the DAVIS during arbitrary motion in a scene with natural textures. This example shows a back-and-forth dominant translation with a short pause in the middle. Events are not displayed to avoid cluttering. A sample of the events produced by the DAVIS in visual odometry applications is shown in Fig. G.2

they operate asynchronously, transmitting only the information conveyed by brightness changes in the scene ("events"), at the time they occur with microsecond resolution. Event-driven algorithms have been developed to provide initial solutions to some robotics problems such as pose tracking [102, 101], visual odometry [26], Simultaneous Localization and Mapping (SLAM) [145, 144]. However, some of these approaches used additional sensors, such as depth sensors [26, 144], or were developed for high-contrast scenes [102, 101, 26].

The Dynamic and Active-pixel Vision Sensor (DAVIS) [19] has been introduced very recently (2014). It is an integrated sensor comprising a conventional frame-based camera and an asynchronous event sensor. This novel hybrid sensor calls for new methods that exploit the combined advantages of event and frame sensors to yield better solutions for robotics problems than those provided by each sensor individually. Such new methods must address the challenges that this sensor poses: it has a complicated analog circuitry, with non-linearities and multiple biases that can change the sensitivity of the pixels, and other dynamic properties, which unfortunately make the frames and events highly susceptible to noise.

Contribution In this paper, we present a low-latency visual odometry algorithm for the DAVIS sensor using event-based feature tracks. We make use of both the frames and the events provided by the sensor. More specifically, we achieve visual odometry using the geometric information conveyed by edge-like features that are adapted to the DAVIS characteristics and represent natural brightness patterns in the scene. Features are first detected using the frames and then tracked asynchronously using the events (see Fig. G.1). Next, they are fed to an event-based visual odometry (VO) algorithm that computes a local probabilistic 3D map of the scene and tracks the 6 degree-of-freedom

(DOF) pose of the sensor by robust reprojection error minimization. Pose updates are event-based, thus preserving the asynchronous and low-latency nature of the event data.

Outline The remainder of the paper is organized as follows. Section G.2 describes the sensor used. Section G.3 reviews related literature on event-based feature tracking and event-driven motion estimation methods. Our approaches to feature tracking (2D) and visual odometry (3D) are described in Sections G.4 and G.5, respectively, and they are empirically evaluated in Section G.6. Conclusions are highlighted in Section G.7.

G.2 The Dynamic and Active-pixel Vision Sensor

The first event-based sensor, called the Dynamic Vision Sensor (DVS) [77], became commercially available in 2008. The DAVIS [19] is a novel vision sensor combining a conventional frame-based camera and a DVS in the same array of pixels. The global-shutter frames provide absolute illumination on demand, whereas the event sensor responds asynchronously to pixel-level brightness changes, independently for each pixel. If $I(t)$ is the illumination sensed at pixel (x, y), an event is triggered if the relative brightness change exceeds a global threshold. More specifically, an event is a tuple $e = (x, y, t, p)$ that conveys the spatio-temporal coordinates (x, y, t) and sign (i.e., polarity $p = \pm 1$) of the brightness change. Events are time-stamped with microsecond resolution and transmitted asynchronously when they occur and with very low latency (microseconds). The DAVIS has a very high dynamic range (130 dB) compared with the 70 dB of high-quality, traditional image sensors. The low latency, the high temporal resolution, and the very high dynamic range make the DAVIS extremely advantageous for future robotic applications.

A visualization of the DAVIS output is shown in Fig. G.2. The spatial resolution of the DAVIS is 240×180 pixels. This is still small compared to the spatial resolution of state-of-the-art conventional cameras. Newer sensors, such as the color DAVIS [75] will have higher spatial resolution (640×480 pixels), thus overcoming current limitations.

Optically, the lenses mounted on the DAVIS are the same as those mounted on conventional cameras. Having the grayscale and DVS pixels perfectly aligned in the DAVIS simplifies camera calibration, an essential stage in robotics applications. State-of-the-art algorithms for conventional cameras can be applied on the frames alone to calibrate the sensor.

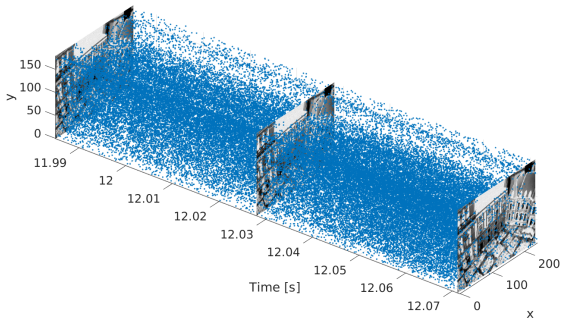


Figure G.2: Typical output of the DAVIS in a visual odometry scenario: as the sensor moves through the scene, it acquires both frames (at low frame rates) and events (asynchronously and fast). Thousands of events are triggered in the time between two frames since, due to the sensor's motion, intensity changes occur at all pixels.

G.3 Related Work

We first review the related works on event-based feature tracking and then on event-based motion estimation.

G.3.1 Event-based Feature Detection and Tracking

Feature detection and tracking methods for frame-based cameras are well established. However, they cannot track in the blind time between consecutive frames, and are expensive because they process information from all pixels, even in the absence of motion in the scene. Conversely, event-based cameras acquire only relevant information for tracking and respond asynchronously, thus, filling the blind time between consecutive frames. Event-based cameras are particularly suitable for applications in motion analysis and high-speed control [35].

Early event-based feature trackers were very simple and focused on demonstrating the low-latency and low-processing requirements of event-driven systems, hence they tracked moving objects as clustered blob-like sources of events [80, 35]. The high-speed advantage of event cameras was also shown in [31] for a pencil-balancing robot. Tracking of the pencil was performed using a fast event-based Hough transform. Tracking of large contrast polygonal shapes was demonstrated in [108], where an event-based Iterative Closest Point (ICP) algorithm was able to track a black polygonal microgripper on a white background. The method allows for planar rigid-body transformations of the target shape and uses a nearest-neighbor strategy to match incoming events to the target shape. Tracking of complex shapes has been recently presented in [109] for the ATIS sensor [116]. The method continuously estimates the geometric transformation between the model and the events representing the object using a gradient descent update. It can handle isometries and mild affine distortions. Tracking the translations

of arbitrary user-defined shapes ("kernels") has also been presented in [72]. Rotational and scaling distortions of a kernel are partially addressed by comparing the events against a collection of rotated and scaled kernels.

All previous methods require a priori knowledge or user input to determine the objects to track. The method in [29] does not detect and track user-defined objects but lower-level primitives, such as corner events defined by the intersection of two moving edges, which are obtained by fitting planes in the space-time stream of events.

The method of [81] uses both the events and frames to detect and track objects. The events are used to track clusters and generate regions of interest, while the frames serve for foreground-background separation using Convolutional Neural Networks (CNN). Since this classification is only applied to the regions of interest, a speedup factor of 70 is reported. The method requires training data and is only suitable for tracking few, large objects in the scene. For visual odometry, we are rather interested in tracking many local features whose position can be precisely determined.

Recently, we presented a hybrid method for feature detection and tracking for the DAVIS [138]. The method first detects and extracts features in the frames and then tracks them using only the events. In the present paper, we improve [138] by (*i*) taking into account the observation that nearby pixels typically observe events at roughly the same time and (*ii*) introducing a tracking refinement step that works on a slower timescale to avoid drift. Furthermore, we improve the tracking speed and add dynamic reinitialization of new features (e.g., in new areas of the scene or when features are lost).

G.3.2 Event-based Motion Estimation

Event-based cameras have been used for robotics applications such as ego-motion estimation and visual odometry. One of the first works in this area is [145], where an event-based particle filter was used for robot self-localization. The VO system was limited to planar motion, 2D reference maps with very high contrast, and known scene depth. In the experiments, they used an upward-looking DVS mounted on a ground robot moving at low speed. The method was extended in [144] to a 3D SLAM system that requires an RGB-D sensor operating in parallel with the DVS.

In our previous work [26], an RGB-D camera was attached to the DVS to estimate the relative displacement between the current event and the previous frame of the camera. However, the system was developed for planar 3-DOF motions and for scenes with very high contrast.

In another work, we presented an event-based algorithm to track the 6-DOF pose of the DVS during high-speed motions in a known environment [102]. However, the method

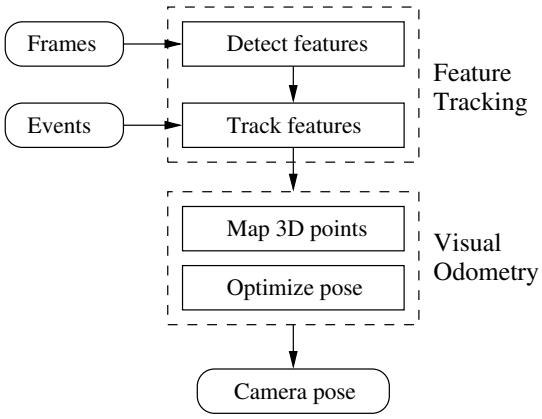


Figure G.3: Visual Odometry system: we track features using events and frames, and then recover the 3D structure of the scene and the DAVIS' pose.

was meant for artificial, B&W line-based maps; indeed, the system estimated the pose through minimizing the point-to-line reprojection error.

Tracking the 3D orientation of the DVS and simultaneously using the event stream to generate high-resolution panorama images of natural scenes was presented in [66, 32]. However, the system was restricted to rotational motions, and, thus, did not account for translation or depth.

None of the previous event-based motion estimation methods is based on tracking complex, natural features in the event stream. This is the approach that we develop in this work, as we show next.

G.4 Feature Detection and Tracking with the DAVIS

The overall system workflow is shown in Fig. G.3. The feature tracking module builds upon our previous work [138], which exploits the absolute brightness information provided by the frames to detect and extract features that are tracked using the event data, as illustrated in Fig. G.4. To make this paper self-contained, we summarize the two main steps, feature detection and tracking, in the next two subsections and propose improvements in Section G.4.3.

G.4.1 Feature Detection using the Frames

Since large contrast edges of moving parts of the scene trigger events more often than low-textured regions, we focus on this dominant information to devise suitable features to track. Moreover, we track only distinctive edge patterns that do not suffer from the aperture problem. Hence, we use the absolute brightness frames of the DAVIS to detect

(a) (b) (c)

Figure G.4: Feature detection and tracking. (a) Frame with centers of detected features (green crosses). (b) (1st zoom) edge map and square patches defining the features (in purple). (c) (2nd zoom) point sets used for feature tracking: model point set (in purple) and data point set (in blue).

Algorithm 2 High temporal resolution tracking

Feature detection:
- Detect corner points on the frame (Harris detector).
- Run Canny edge detector (returns a binary image, 1 if edge pixel, 0 otherwise).
- Extract local edge-map patches around corner points, and convert them into model point sets.

Feature tracking:
- Initialize the data point set and 2D histograms per patch.

for each incoming event **do**
- Update the corresponding data point set and histograms

for each corresponding data point set **do**
- Estimate the registration parameters between the data and the model point sets (weighted ICP).
- Update registration parameters of the model points.

Every M_2 events, compare spatial histograms, compute and apply the best shift to mitigate drift.

both edges (Canny's method [23]) and corners (Harris detector [58]). Around the most salient and best distributed corners in the frame (Fig. G.4a), we use the edge pixels to define patches that mark the locations of the dominant sources of events (Fig. G.4b). The patches are converted into binary masks that indicate the presence (1) or absence (0) of an edge. These patches resemble the customized kernels in [72]. The binary masks define the target shapes to be tracked as 2D point sets, called "model point sets" (Fig. G.4c). These steps are summarized in the first part of Algorithm 2.

All patches are square and have the same size (Fig. G.4b), which is an adjustable parameter, but it is straightforward to extend the method to consider different patch sizes.

Our method does not require frames to be provided at a constant rate since they are

only used to initialize features. Frames can be acquired on demand to replace lost features.

G.4.2 Feature Tracking using the Events

Detected features are tracked using the event stream. The input to the event-based tracking algorithm consists of multiple, local model point sets. The tracking strategy is summarized in the second part of Algorithm 2.

Sets of Active Events

For every feature i, we define a data point set of the same size N_p^i as the model point set. Data point sets consist of subsets of the incoming events: an event is inserted in the i-th data point set if its coordinates are inside the i-th patch. A data point set defines the active set of events that are relevant for the registration of the corresponding feature (Fig. G.4c). Data point sets are continuously updated: every incoming event replaces the oldest one in the set, and then the registration iteration proceeds. This strategy is event-based, which means that the registration parameters of the tracked feature are updated every time an incoming event is considered relevant for the feature under consideration. The algorithm is asynchronous by design and it can process multiple features simultaneously.

Registration

Registration is carried out by minimization of a weighted distance between the model (feature) and the data point sets (events), $\mathbf{m}_i$ and $\mathbf{p}_i$, respectively:

$$\arg\min_{\mathtt{R},\mathbf{t}} = \sum_{(\mathbf{p}_i,\mathbf{m}_i)\in\text{Matches}} b_i \left\|\mathtt{R}\mathbf{p}_i + \mathbf{t} - \mathbf{m}_i\right\|^2, \tag{G.1}$$

where a Euclidean transformation $(\mathtt{R}, \mathbf{t})$ is assumed (this is enough since the features do not significantly deform in the time between events), and the weights b_i take into account outlier rejection and simultaneous events due to edge structure (Section G.4.3). The algorithm that minimizes (G.1) is a variation of the Iterative Closest Point (ICP) algorithm [12]. It yields the tracking update rule and is directly linked to our choice of feature representation. Each iteration of the algorithm has three stages: first, candidate matches are established, then the geometric transformation is estimated, and, finally, the transformation is applied to the model point set. The operation proceeds until the error difference between two consecutive iterations is below a certain threshold. Matches are established according to the minimum distance criterion, and those above a certain threshold are discarded; hence the method can handle outlier events, as those produced by noise.

Due to the high temporal resolution of the DAVIS, the transformation between consecutive events (in the same feature) is close to the identity and, therefore, our method yields good results even after a single iteration.

G.4.3 Tracking Improvements

Moving edges produce simultaneous events at neighboring locations

Tracking accuracy is improved by incorporating in the registration criterion the fact that our features are based on edges, which are not isolated points but form connected structures that normally trigger events in neighboring locations at roughly the same time. We do so by using weights b_i in (G.1) proportional to the number of events, out of the last $N_p^i/4$ of them, that fall in the 3×3 pixel neighborhood of the current event.

Tracking refinement based on local 2D histograms of events

We supplement weighted ICP with local 2D histograms designed to improve long-term tracking. Events are accumulated into patch-size histograms over longer times than those used in ICP, which makes them more robust to noise than the point sets. There are two histograms per feature: H_1 over the first M_1 events and a moving histogram H_2 over the last M_2 events. For well-tracked and rotation-compensated features, both histograms look almost identical, thus effectively filtering out noisy events. Otherwise, feature drift is detected as a shift of H_2 with respect to H_1. Every M_2 events, histograms are compared in search for shifts $\mathbf{s} = (o_x, o_y)^\top$ in the range ± 3 pixels in each dimension. The comparison metric is histogram intersection, given by the sum of the minimum of the two histogram values:

$$d(H_1, H_2, \mathbf{s}) = \sum_{\mathbf{x}} \min(H_1(\mathbf{x}), H_2(\mathbf{x} + \mathbf{s})), \tag{G.2}$$

where $\mathbf{x} = (x, y)^\top$ iterates over the patch domain. The shift $\mathbf{s}$ with the largest intersection is applied to the feature, if it is larger than a given threshold. Histogram lengths pose a trade-off: the larger they are, the more event noise is filtered; however, this decreases the reaction speed of the algorithm and it can also yield blurred histograms for fast drifting features. The histogram lengths M_1 and M_2 are multiples of the patch size N; a good choice for M_2 is $5N$, while M_1 is set larger than M_2 to ensure a good initial histogram. Figure G.5 shows a sequence of histograms with $M_2 = 5N$. The benefit of this technique on tracking and visual odometry is demonstrated in Section G.6.1.

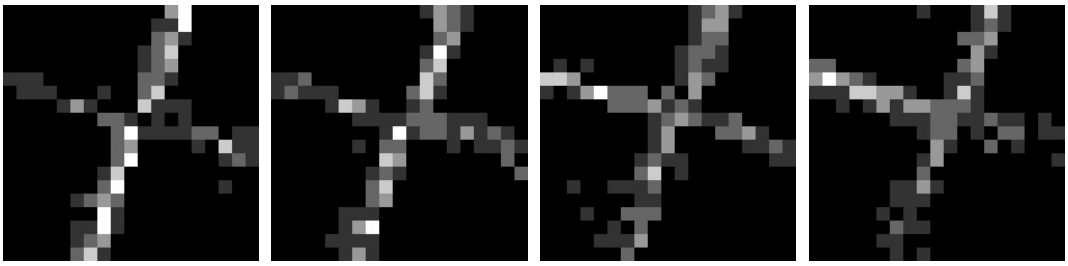


Figure G.5: From left to right: histogram H_1 (initial) and three instances of histogram H_2 as new events arrive.

G.5 Visual Odometry

The proposed visual odometry (VO) algorithm in Fig. G.3 uses the 2D geometric information provided by the feature tracks to estimate the 3D structure of the scene and the location of the DAVIS. These two operations (localization and mapping) are performed in a tightly interleaved manner. We use depth-filters [50] to estimate the scene structure in a Bayesian way, and since we track in the order of one hundred features, the resulting probabilistic map is a sparse representation of the scene. The camera motion is tracked by minimization of a weighted reprojection error using the Gauss-Newton method, which is very fast since the motion between two events is almost zero. Both operations are adapted from the Semi-direct Visual Odometry (SVO) algorithm [50].

G.5.1 3D Mapping using Depth Filters

During mapping, we estimate the depth of 2D features for which the corresponding 3D point is not yet known. The depth estimate of a feature is modeled with a probability distribution that is updated in a Bayesian framework (see Fig. G.6) [50]. This is known as a depth-filter. When a depth-filter has converged, that is, when the variance of the distribution becomes small enough, a new 3D point is inserted in the map at the converged depth and it is immediately used for pose tracking.

More specifically, depth-filters are initialized with a high uncertainty and set to the mean depth of the scene. Feature tracks provide depth measurements that are processed by the filter. Each measurement $\tilde{d}_i^k$ (k-th observation of the i-th feature) is obtained by triangulation of the current feature location and its first detection, using the relative camera pose $\mathbf{T}_{r,k}$ (see Fig. G.6). $\tilde{d}_i^k$ is modeled using a Gaussian + Uniform mixture [143]: good measurements are normally distributed around the true depth d_i, with variance τ_i^2, while outliers are uniformly distributed in the known range $[d_i^{\min}, d_i^{\max}]$,

$$p(\tilde{d}_i^k | d_i, \rho_i) = \rho_i \mathcal{N}(\tilde{d}_i^k | d_i, \tau_i^2) + (1 - \rho_i)\mathcal{U}(\tilde{d}_i^k | d_i^{\min}, d_i^{\max}),$$

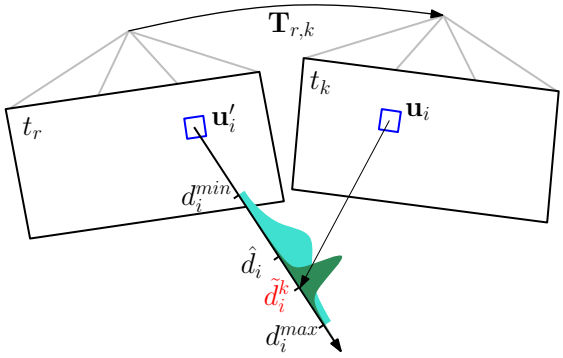


Figure G.6: Depth-filter update for a new measurement $\tilde{d}_i^k$, at current time t_k, of a feature that was extracted at reference time t_r. Over time the uncertain distribution (cyan) becomes narrower for an inlier (green). Image courtesy of [50].

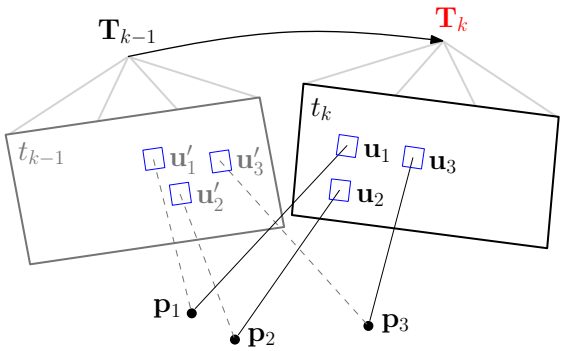


Figure G.7: The Gauss-Newton optimizer finds the new pose $\mathbf{T}_k$ at time t_k from 2D-3D feature correspondences $\mathbf{u}_i \leftrightarrow \mathbf{p}_i$ and the initial guess $\mathbf{T}_{k-1}$. Image courtesy of [50].

where ρ_i is the inlier probability. Well-tracked features are those with ρ_i close to 1. Further details on the filter update equations can be found in [143], however, note that we use inverse depth coordinates, as in [50].

G.5.2 Pose Tracking by Reprojection Error Minimization

Given a sparse map of the scene, we obtain the current camera pose $\mathbf{T}_k$ by minimizing the reprojection error:

$$\mathbf{T}_k = \arg\min_{\mathbf{T}} \tfrac{1}{2} \sum_i w_i \|\mathbf{u}_i - \pi(\mathbf{T}, \mathbf{p}_i)\|^2, \tag{G.3}$$

where $\mathbf{u}_i$ and $\mathbf{p}_i$ are the 2D and 3D positions of the i-th feature, as illustrated in Fig. G.7, w_i are weights, and π projects 3D world points into the camera frame. This is solved iteratively with the Gauss-Newton method. For robustness against outliers, we use

the bell-shaped Tukey weight function

$$w_i = \begin{cases} \left(1 - \frac{x^2}{b^2}\right)^2 & |x| \leq |b|, \\ 0 & \text{otherwise}, \end{cases} \tag{G.4}$$

with $x = \|\mathbf{u}_i - \pi(\mathbf{T}, \mathbf{p}_i)\|$ and $b = 5$ (pixels). Additionally we have found that by multiplying w_i by the inlier probability ρ_i and the normalized feature age (the number of events that fall into the patch), the results are significantly better. Features that are well-tracked over a long time are given the highest weight, while features that lose track are usually removed due to a large reprojection error.

G.5.3 Bootstrapping

Pose optimization relies on the availability of a map, and mapping relies on the availability of pose information. But neither of them are available at the beginning, hence we need to provide an initial map and pose estimate. We use two-view bootstrapping with a minimum mean disparity between the features. The relative camera pose is calculated from 2D point correspondences with the five-point algorithm for the essential matrix [110] and RANSAC [48]. This requires an initial camera motion with a translational component.

G.6 Experiments

We evaluated the performance of the event-based VO system on several scenes with natural textures, rich in brightness changes of different magnitudes (e.g., Fig. G.8). The resulting camera trajectory is compared against those acquired by a motion-capture system and a frame-based VO algorithm based on the state of the art [50]. No constraints were placed on the sensor's motion: the DAVIS was freely moved by hand through the scene.

G.6.1 Feature Tracking

A space-time visualization of the trajectories described by the tracked features in the image plane is displayed in Fig. G.1. Qualitatively, it shows that the tracked features' trajectories have a coherent motion.

We evaluate our event-based feature tracker on two different scenes: a checkerboard-like scene (because it offers well-localized features) and a natural scene (with less localized features, Fig. G.8). The results are reported in Fig. G.9 and Fig. G.10, respectively, in terms of tracking error vs. time. The tracking error is computed against ground truth. Ground truth was generated using a frame-based Lucas-Kanade tracker [85] and

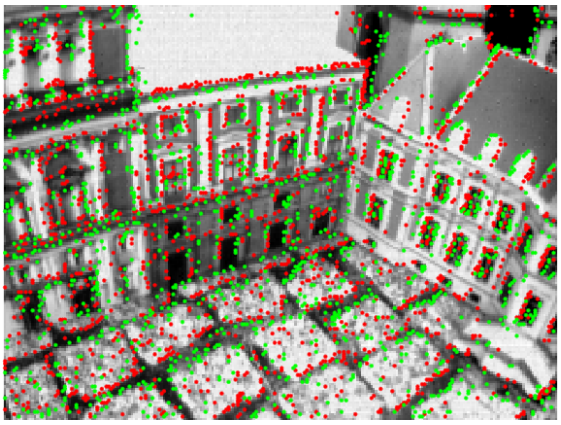


Figure G.8: Sample output of the DAVIS: frame with overlaid events, colored according to polarity (positive in green, negative in red), from a 0.5 ms interval.

linearly interpolating the feature motion in the time interval between frames. Features were detected in the first frame ($t = 0$) and then tracked over the entire sequence using only events. In the checkerboard-like scene (Fig. G.9), the mean tracking error is 1.5 pixels. In the natural scene (Fig. G.8), the tracking accuracy gracefully degrades, yielding a mean tracking error of 2.5 pixels. This degradation results from two causes: (*i*) we do not model many of the non-linearities of the DAVIS, such as non-white noise and other dynamic properties; (*ii*) the detected features are based on a binary edge-map of the scene (resulted from the Canny detector), but such binary map is an exact representation of the underlying grayscale scene only if the contrast is very large. In natural scenes, edges can have all sort of different magnitudes, but our features still track the most dominant ones. We used patches of 19×19 pixels, which were experimentally proven to be best for a broad class of scenes.

The positive effect of the feature refinement technique described in Section G.4.3 is shown in Fig. G.11. Feature refinement increases the duration of the feature tracks (called "feature age"). This has also a positive effect on the performance of the VO algorithm since points that are tracked for longer times imply higher VO accuracy.

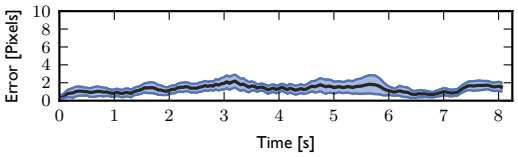


Figure G.9: Feature tracking error of our event-based algorithm on the checkerboard-like scene. The mean tracking error of all features is marked in black. The blue bands around the mean indicate the ± 1 standard-deviation confidence interval. The overall mean error (in black) is 1.5 pixels.

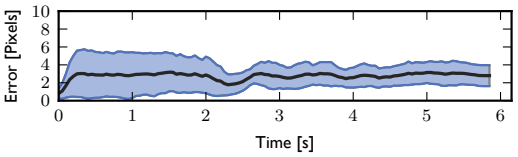


Figure G.10: Feature tracking error of our event-based algorithm on the natural scene in Fig. G.8. The mean tracking error of all features is marked in black. The blue bands around the mean indicate the ± 1 standard-deviation confidence interval. The overall mean error (in black) is 2.5 pixels.

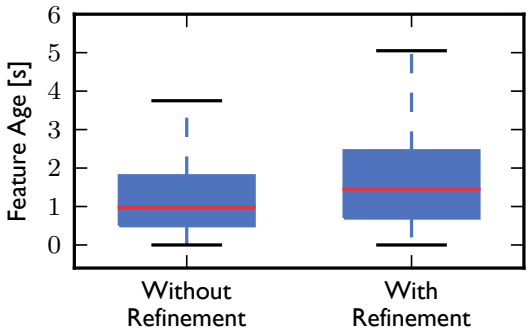


Figure G.11: Long-term tracking improves by using local spatial histograms of events. With them, the "feature age" distribution shifts toward higher values; e.g., the median increases from 1.0 s to 1.5 s.

G.6.2 Visual Odometry

The VO system tracks a constant number of features (120) that are chosen to be well distributed in the image plane by means of a grid/binning strategy. Using fewer than 100 features does not yield good results. New features are initialized when features are lost or leave the field of view (FOV) of the DAVIS.

Figs. G.12 and G.13 show the VO results on two sequences. The figures show the trajectory produced by our VO algorithm, the ground truth (motion-capture system), and the trajectory produced by a frame-based solution, as well as the corresponding errors in position and orientation (with respect to ground truth). The position error is given by the Euclidean distance between camera locations. The orientation error is given by the angle of the relative rotation between camera reference frames, which is the geodesic distance in SO(3) [62]. In Experiment 1 (Fig. G.12), the average position errors of our method and the frame-based one are 16 and 6 mm, respectively. Since the mean scene depth is 40 cm, this corresponds to relative errors of 4.0 % and 1.5 %, respectively. In Experiment 2 (Fig. G.13), these errors are 30 and 11 mm (7.5 % and 2.8 % of the mean scene depth), respectively. Overall, the quality of the estimate produced by our event-based algorithm is comparable to that of the frame-based solution, but provides low-latency pose updates since it preserves the event-based nature of the data. These results are very promising and represent a first step towards a fully integrated frame-plus-events feature tracking and VO solution with this novel sensor in natural scenes and in 6-DOF motions, which are challenging conditions that have not been addressed in previous work.

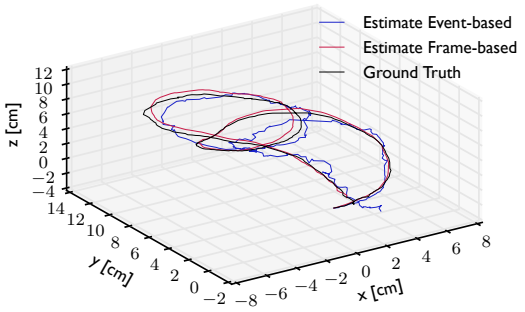
G.6.3 Runtime Analysis

The runtime of event-based algorithms depends on the event rate, which itself depends on several factors: the apparent speed of moving objects in the scene, the amount of texture and edges, the sensor parameters (bias configuration), etc. Roughly speaking,

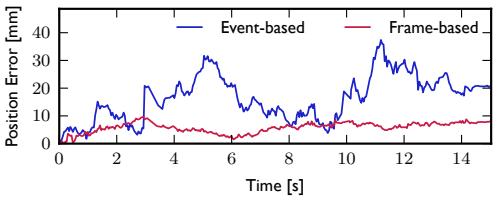
the DAVIS generates 10^5–10^6 events/s for normal motions. For faster motion, it is in the order of a few millions. We tested the implementation on a laptop with Intel Core i7-4710MQ CPU @ 2.50GHz with 8GB RAM. The C++ code runs completely single threaded. Our algorithm is able to process 160 kevents/s on average. The performance analysis is shown in Fig. G.14. Running ICP on every incoming event is excessive since an event does not have a large influence. We experimentally found that running ICP every $N/3$ events (N being the size of the point sets) speeds up the VO algorithm by a factor of 6 while its accuracy is preserved. Running ICP only every N events increases the error. In spite of this speed-up, a significant portion of the computational time (58.5%, see Fig. G.14) is spent in an off-the-shelf ICP library [115] that can match point sets in arbitrary dimensions. By using a customized implementation, runtime could be improved.

G.7 Conclusion

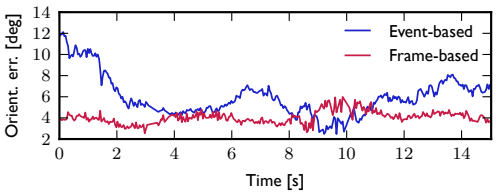
We have developed a low-latency, event-based visual odometry algorithm adapted to the characteristic of the DAVIS, a prototype sensor with great potential for robotics, which combines a conventional camera and an event-based sensor in the same pixel array. Our method extracts visual features in the frames and tracks them asynchronously using the events. Features are designed to be trackable using only the events and are sufficiently generic to be applicable to non-structured environments. We used two cooperative techniques to achieve event-based tracking: a weighted point-set minimization for short-term tracking and comparison of spatial histograms of events for long-term tracking. Feature tracks were used to infer 3D quantities, using probabilistic depth-filters for mapping and robust reprojection error minimization for pose tracking. We demonstrated successful tracking and VO performance of a moving DAVIS in 6-DOF and in scenes with natural textures, which are challenging conditions that have not been previously addressed in the literature.



(a) 3D view of camera trajectories

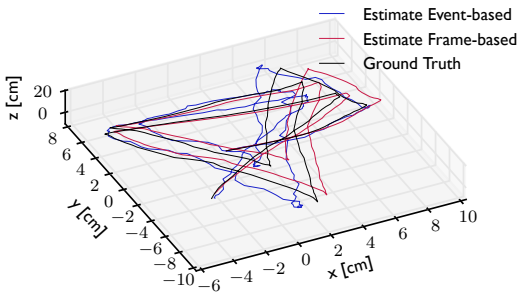


(b) Position error of the estimated trajectories (event-based and frame-based) with respect to ground truth. The mean scene depth is 40 cm.

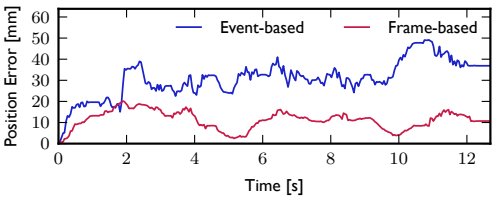


(c) Orientation error of the estimated trajectories (event-based and frame-based) with respect to ground truth.

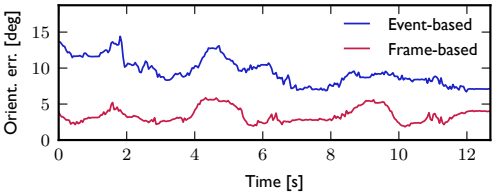
Figure G.12: *VO Experiment 1.* Comparison of DAVIS trajectories estimated by our event-based VO algorithm, a frame-based solution, and ground truth (motion-capture system).



(a) 3D view of camera trajectories



(b) Position error of the estimated trajectories (event-based and frame-based) with respect to ground truth. The mean scene depth is 40 cm.



(c) Orientation error of the estimated trajectories (event-based and frame-based) with respect to ground truth.

Figure G.13: *VO Experiment 2.* Comparison of DAVIS trajectories estimated by our event-based VO algorithm, a frame-based solution, and ground truth (motion-capture system).

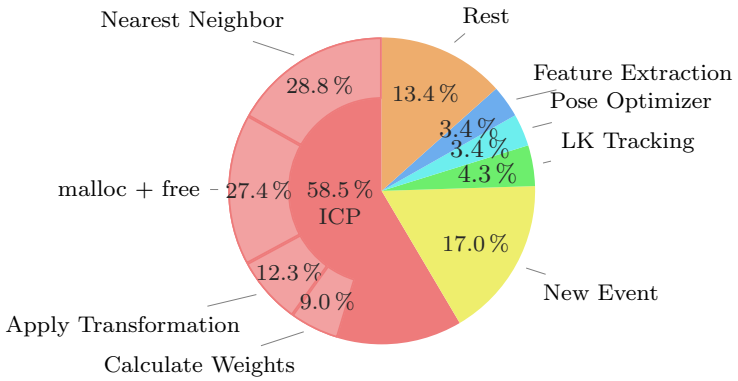


Figure G.14: Runtime analysis of the VO system in Fig. G.3, calling ICP every $N/3$ incoming events. "New Event" checks whether a new event is within a patch and updates the target point set accordingly. "LK Tracking" is not part of our algorithm and is only used for comparison. Feature extraction and pose tracking are very efficient. Percentages within ICP are relative to the aggregated cost (58.5%).

H Event-based Dense Tracking

This chapter is a reprint of the article currently under revision as:

G. Gallego, J. E. A. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza. "Event-based, 6-DOF Camera Tracking for High-Speed Applications". In: *IEEE Trans. Pattern Anal. Machine Intell.* (2017). under review

Event-based, 6-DOF Camera Tracking for High-Speed Applications

Guillermo Gallego, Jon E. A. Lund, Elias Mueggler, Henri Rebecq, Tobi Delbruck and Davide Scaramuzza

Abstract — In contrast to standard cameras, which produce frames at a fixed rate, event cameras respond asynchronously to pixel-level brightness changes, thus enabling the design of new algorithms for high-speed applications with latencies of microseconds. However, this advantage comes at a cost: because the output is composed by a sequence of events, traditional computer-vision algorithms are not applicable, so that a new paradigm shift is needed. We present an event-based approach for ego-motion estimation, which provides pose updates upon the arrival of each event, thus virtually eliminating latency. Our method is the first work addressing and demonstrating event-based pose tracking in six degrees-of-freedom (DOF) motions in realistic and natural scenes, and it is able to track high-speed motions. The method is successfully evaluated in both indoor and outdoor scenes with significant depth variation, and under motions with excitations in all 6-DOFs.

H.1 Introduction

Event cameras [8, p.77], such as the Dynamic Vision Sensor (DVS [77]), are biologically inspired sensors that overcome many limitations of traditional cameras: they respond very fast (within microseconds) to brightness changes, have a very high dynamic range (120 dB vs 60 dB of standard cameras), and require low power and bandwidth (20 mW vs 1.5 W of standard cameras). Such advantages makes these sensors very attractive for low-powered and/or high-speed applications. However, because they convey the visual information in a radically different way than standard cameras (they do not

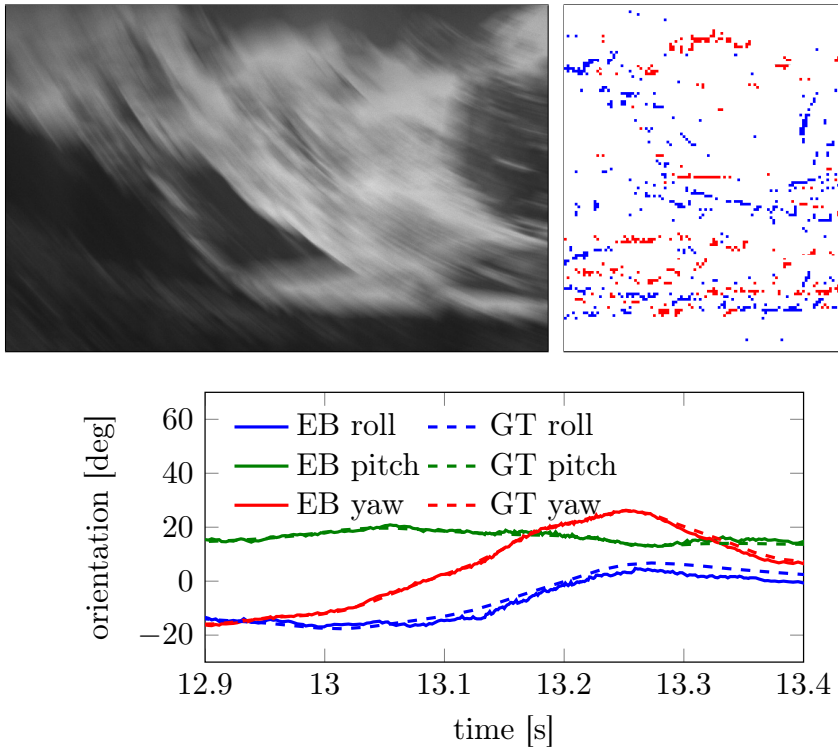


Figure H.1: High-speed motion sequence. Top left: image from a standard camera, suffering from blur due to high-speed motion. Top right: set of asynchronous DVS events in an interval of 3 milliseconds, colored according to polarity. Bottom: estimated poses using our event-based (EB) approach, which provides low latency and high temporal resolution updates. Ground truth (GT) poses are also displayed.

provide grayscale values but only changes in intensity and the output is composed by a sequence of asynchronous events rather than frames), computer-vision algorithms that are conceived for conventional frame-based cameras do not work on event data. Therefore, new methods must be developed to leverage the advantages of event-driven vision [129].

Previous works on event cameras are still at an early stage of development since event cameras have become commercially available only since 2008 [77]. How to exploit the advantages of event cameras (i.e., high speed, low latency, and high dynamic range) is still an open research problem. The challenges we address in this paper are two: *i*) event-based 6-DOF pose tracking in natural scenes; *ii*) tracking the pose during very fast motions, e.g., where standard cameras suffer from motion blur, as shown in Figure H.1.

We present a novel probabilistic pose-tracking method for event-based vision sensors in a known environment. It is based on Bayesian filtering theory with three key contributions in the way that the events are processed: *i*) event-based pose update,

meaning that the 6-DOF pose estimate is updated every time an event is generated, at *micro*second time resolution, *ii*) the design of a sensor likelihood function using a mixture model that takes into account both the event generation process and the presence of noise and outliers (Section H.4.3), and *iii*) the approximation of the posterior distribution of the system by a tractable distribution in the exponential family that is obtained by minimizing the Kullback-Leibler divergence (Section H.4.4). The result is a filter adapted to the asynchronous nature of the DVS, which also incorporates an outlier detector that weighs measurements according to their confidence for improved robustness of the pose estimation. The approximation of the posterior distribution allows us to obtain a closed-form solution to the filter update equations and has the benefit of being computationally efficient. Localization of the DVS is achieved with respect to reference images (and their poses) of the scene. Our method can handle arbitrary, 6-DOF, high-speed motions of the DVS in natural scenes.

The paper is organized as follows: Section H.2 reviews related literature on event-based ego-motion estimation methods. Section H.3 describes the Dynamic Vision Sensor. The probabilistic approach developed for DVS 6-DOF pose tracking is described in Section H.4, and it is empirically evaluated on natural scenes in Section H.5. Conclusion and future work are highlighted in Section H.6.

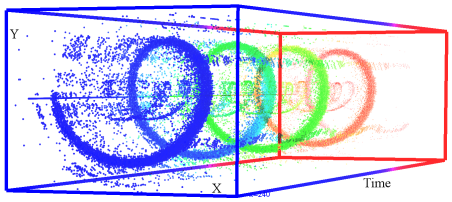
H.2 Related work on Event-based Ego-Motion Estimation

The first work on pose tracking with a DVS was presented in [145]. The system design, however, was limited to slow planar motions (i.e., 3 DOF) and planar scenes parallel to the plane of motion consisting of artificial B&W line patterns. The method was extended to 3-D in [144] but relied on an external RGB-D sensor for depth estimation. However, a depth sensor introduces the same bottlenecks that exist in standard frame-based systems: depth measurements are outdated for very fast motions, and the depth sensor is still susceptible to motion blur.

In our previous work [26], a standard grayscale camera was attached to a DVS to estimate the small displacement between the current event and the previous frame of the standard camera. The system was developed for planar motion and artificial B&W striped background. This was due to the sensor likelihood being proportional to the magnitude of the image gradient, thus favoring scenes where large brightness gradients are the source of most of the event data. Because of the reliance on a standard camera, the system was again susceptible to motion blur and therefore limited to slow motions.

An event-based algorithm to track the 6-DOF pose of a DVS alone and during very high-speed motion was presented in [102]. However, the method was developed specifically for artificial, B&W line-based maps. Indeed, the system worked by minimizing the point-to-line reprojection error.

(a) The Dynamic Vision Sensor (DVS).



(b) Visualization of the output of a DVS (event stream) while viewing a rotating scene, which generates a spiral-like structure in space-time. Events are represented by colored dots, from red (far in time) to blue (close in time). Event polarity is not displayed. Noise is visible by isolated points.

Figure H.2: Event-based camera.

Estimation of the 3-D orientation of a DVS to generate high-resolution panoramas of natural scenes was presented in [66]. However, the system was restricted to rotational motions, and, thus, did not account for translation and depth.

Contrarily to all previous works, the approach we present in this paper tackles full 6-DOF motions, does not rely on external sensors, can handle arbitrary fast motions, and is not restricted to specific texture or artificial scenes.

H.3 Event-based cameras. The Dynamic Vision Sensor (DVS)

Event-based vision constitutes a paradigm shift from conventional (e.g., frame-based) vision. In standard cameras, pixels are acquired and transmitted simultaneously at fixed rates; this is the case of both global-shutter or rolling-shutter sensors. Such sensors provide little information about the scene in the "blind time" between consecutive images. Instead, event-based cameras such as the DVS [77] (Figure H.2a) have independent pixels that respond asynchronously to relative contrast changes. If $I(\mathbf{u}, t)$ is the intensity sensed at a pixel $\mathbf{u} = (x, y)^\top$ of the DVS, an event is generated if the temporal visual contrast (in log scale) exceeds a nominal threshold C_{th}:

$$\Delta \ln I := \ln I(\mathbf{u}, t) - \ln I(\mathbf{u}, t - \Delta t) \gtrless C_{\text{th}}, \tag{H.1}$$

where Δt is the time since the last event was generated at the same pixel. Different thresholds may be specified for the cases of contrast increase (C_{th}^+) or decrease (C_{th}^-). An event $e = (x, y, t, p)$ conveys the spatio-temporal coordinates and sign (i.e., polarity) of the brightness change, with $p = +1$ (ON-event: $\Delta \ln I > C_{\text{th}}^+$) or $p = -1$ (OFF-event:

$\Delta \ln I < C_{\text{th}}^{-}$). Events are time-stamped with microsecond resolution and transmitted asynchronously when they occur, with very low latency. A sample output of the DVS is shown in Figure H.2b. Another advantage of the DVS is its very high dynamic range (120 dB), which notably exceeds the 60 dB of high-quality, conventional frame-based cameras. This is a consequence of events triggering on log-intensity changes (H.1) instead of absolute intensity. The spatial resolution of the DVS is 128×128 pixels, but newer sensors, such as the Dynamic and Active-pixel VIsion Sensor (DAVIS) [19], and color DAVIS (C-DAVIS) [75] will have higher resolution (640×480 pixels), thus overcoming current limitations.

H.4 Probabilistic approach

Consider a DVS moving in a known static scene. The map of the scene is described by a sparse set of reference images $\{I_l^r\}_{l=1}^{N_r}$, poses $\{\boldsymbol{\xi}_l^r\}_{l=1}^{N_r}$, and depth map(s). Suppose that an initial guess of the location of the DVS in the scene is also known. The problem we face is that of exploiting the information conveyed by the event stream to track the pose of the DVS in the scene. Our goal is to handle arbitrary 6-DOF, high-speed motions of the DVS in realistic (i.e., natural) scenes.

We design a robust filter combining the principles of Bayesian estimation, posterior approximation, and exponential family distributions with a sensor model that accounts for outlier observations. In addition to tracking the kinematic state of the DVS, the filter also estimates some sensor parameters automatically (e.g., event triggering threshold C_{th}) that would otherwise be difficult to tune manually. [1]

The outline of this section is as follows. First, the problem is formulated as a marginalized posterior estimation problem in a Bayesian framework. Then, the motion model and the measurement model (a robust likelihood function that can handle both good events and outliers) are presented. Finally, the filter equations that update the parameters of an approximate distribution to the posterior probability distribution are derived.

H.4.1 Bayesian Filtering

We model the problem as a time-evolving system whose state s consists of the kinematic description of the DVS as well as sensor and inlier/outlier parameters. More specifically,

$$s = (\boldsymbol{\xi}_c, \boldsymbol{\xi}_i, \boldsymbol{\xi}_j, C_{\text{th}}, \pi_m, \sigma_m^2)^\top, \tag{H.2}$$

[1] Today's event-based cameras, such as the DVS [77] or the DAVIS [19], have almost a dozen tuning parameters that are neither independent nor linear.

where $\boldsymbol{\xi}_c$ is the current pose of the DVS (at the time of the event, t in (H.1)), $\boldsymbol{\xi}_i$ and $\boldsymbol{\xi}_j$ are two poses along the DVS trajectory that are used to interpolate the pose of the last event at the same pixel (time $t - \Delta t$ in (H.1)), C_{th} is the contrast threshold, and π_m and σ_m^2 are the inlier parameters of the sensor model, which is explained in Section H.4.3.

Let the state of the system at time t_k be s_k, and let the sequence of all past observations (up to time t_k) be $o_{1:k}$, where o_k is the current observation (i.e., the latest event).

Our knowledge of the system state is contained in the posterior probability distribution $p(s_k|o_{1:k})$, also known as *belief* [139, p.27], which is the marginalized distribution of the smoothing problem $p(s_{1:k}|o_{1:k})$. The Bayes filter recursively estimates the system state from the observations in two steps: prediction and correction. The correction step updates the posterior by:

$$p(s_k|o_{1:k}) \propto p(o_k|s_k)p(s_k|o_{1:k-1}), \tag{H.3}$$

where $p(o_k|s_k)$ is the likelihood function (sensor model) and we used independence of the events given the state. The prediction step, defined by

$$p(s_k|o_{1:k-1}) = \int p(s_k|s_{k-1})p(s_{k-1}|o_{1:k-1})ds_{k-1}, \tag{H.4}$$

incorporates the motion model $p(s_k|s_{k-1})$ from t_{k-1} to t_k.

We incorporate in our state vector not only the current DVS $\boldsymbol{\xi}_c^k$ pose but also the other relevant poses for contrast calculation (poses $\boldsymbol{\xi}_i^k, \boldsymbol{\xi}_j^k$ in (H.2)), so that we may use the filter to partially correct errors of already estimated poses. Past events that are affected by the previous pose are not re-evaluated, but future events that reference back to such time will have better previous-pose estimates.

To have a computationally feasible filter, we approximate the posterior (H.3) by a tractable distribution with parameters η_{k-1} that condense the history of events $o_{1:k-1}$,

$$p(s_k|o_{1:k}) \approx q(s_k; \eta_k). \tag{H.5}$$

Assuming a motion model with slowly varying zero-mean random diffusion, so that most updates of the state are due to the events, the recursion on the approximate posterior becomes, combining (H.3)-(H.5),

$$q(s_k; \eta_k) \approx C\, p(o_k|s_k)q(s_k; \eta_{k-1}) \tag{H.6}$$

for some normalizing constant C. The approximate posterior q is computed by minimization of the Kullback-Leibler (KL) divergence between both sides of (H.6). As tractable distribution we choose one in the exponential family because they are very flexible and have nice properties for sequential Bayes estimation. The KL minimization

gives the update equations for the parameters of the approximate posterior.

H.4.2 Motion model

The diffusion process leaves the state mean unchanged and propagates the covariance. How much process noise is added to the evolving state is determined by the trace of the covariance matrix (sum of the eigenvalues): each incoming event adds white noise to the covariance diagonal, thus increasing its trace, up to some allowed maximum. This works gracefully across many motion speeds. More specifically, we used a maximum standard deviation of 0.03 for poses parametrized in normalized coordinates (with translation in units relative to the mean scene depth), to factor out the metric scale in the diffusion process.

H.4.3 Measurement Model

Here we elaborate on the choice of likelihood function $p(o_k|s_k)$ in (H.6) that is used to model the DVS events. Our contributions are, starting from an ideal sensor model, *i*) to define a dimensionless implicit function based on the contrast residual to measure how well the DVS pose and the a priori information (e.g., a map of the scene) explain an event (Section H.4.3), and *ii*) to build upon such measurement function taking into account noise and outliers, yielding a mixture model for the likelihood function (Section H.4.3).

Ideal Sensor Model

In a noise-free scenario, an event is triggered as soon as the temporal contrast reaches the threshold (H.1). Such a measurement would satisfy $\Delta \ln I - C_{\text{th}} = 0$. For simplicity, let us assume that the polarity has already been taken into account to select the appropriate threshold $C_{\text{th}}^{+} > 0$ or $C_{\text{th}}^{-} < 0$. Defining the measurement function

$$M := \frac{\Delta \ln I}{C_{\text{th}}} - 1, \tag{H.7}$$

the event-generation condition becomes $M = 0$ in a dimensionless formulation. Assuming a prediction of the temporal contrast is generated using the system state, $\Delta \ln I(s_k)$, then (H.7) depends on both the system state and the observation, $M(o_k, s_k)$. More precisely, denoting by

$$\tilde{s} = (\boldsymbol{\xi}_c, \boldsymbol{\xi}_i, \boldsymbol{\xi}_j, C_{\text{th}})^{\top}, \tag{H.8}$$

the part of the state (H.2) needed to compute (H.7), we have $M(o_k, \tilde{s}_k)$. The likelihood function that characterizes such an ideal sensor model is

$$p(o_k|s_k) = \delta(M(o_k, \tilde{s}_k)), \tag{H.9}$$

where δ is the Dirac delta distribution.

All deviations from ideal conditions can be collectively modeled by a noise term in the likelihood function. Hence, a more realistic yet simple choice than (H.9) that is also supported by the bell-shaped form of the threshold variations observed in the DVS [77] is a Gaussian distribution,

$$p(o_k|s_k) = \mathcal{N}(M(o_k, \tilde{s}_k); 0, \sigma_m^2). \tag{H.10}$$

Most previous works in the literature do not consider an implicit measurement function (H.7) or Gaussian model (H.10) based on the contrast residual. Instead, they use explicit measurement functions that evaluate the goodness of fit of the event either in the spatial domain (reprojection error) [145, 102] or in the temporal domain (event-rate error), e.g., image reconstruction thread of [66], assuming Gaussian errors. Our measurement function (H.7) is based on the event-generation process and combines in a scalar quantity all the information contained in an event (space-time and polarity) to provide a measure of its fit to a given state and a priori information. However, models based on a single Gaussian distribution (H.10) are very susceptible to outliers. Therefore, we opt for a mixture model to explicitly account for them, as explained next.

Resilient Sensor Model. Likelihood Function

Based on the empirical observation that there is a significant amount of outliers in the event stream, we propose a likelihood function consisting of a normal-uniform mixture model. This model is typical of robust sensor fusion problems [143], where the output of the sensor is modeled as a distribution that mixes a good measurement (normal) with a bad one (uniform):

$$\begin{aligned} p(o_k|s_k) = {} & \pi_m \, \mathcal{N}(M(o_k, \tilde{s}_k); 0, \sigma_m^2) \\ & + (1 - \pi_m) \, \mathcal{U}(M(o_k, \tilde{s}_k); M_{\min}, M_{\max}), \end{aligned} \tag{H.11}$$

where π_m is the inlier probability (and $(1 - \pi_m)$ is the outlier probability). Inliers are normally distributed around 0 with variance σ_m^2. Outliers are uniformly distributed over a known interval $[M_{\min}, M_{\max}]$. The measurement parameters σ_m^2 and π_m are considered unknown and are collected in the state vector s_k to be estimated.

To evaluate $M(o_k, \tilde{s}_k)$, we need to compute the contrast $\Delta \ln I(\tilde{s}_k)$ in (H.7). We do so

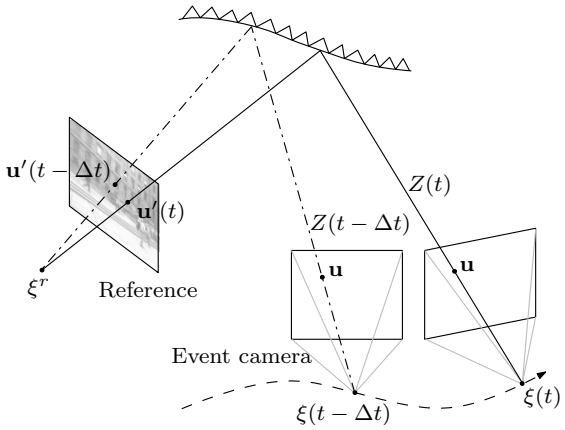


Figure H.3: Computation of the contrast (measurement function) by transferring events from the DVS to a reference image. For each event, the predicted contrast (H.13), $\Delta \ln I$, used in the measurement function (H.7) is computed as the log-intensity difference (as in (H.1)) at two points on the reference image I^r: the points (H.12) corresponding to the same pixel $\mathbf{u}$ on the event camera, at times of the event (t_k and $t_k - \Delta t$).

based on a known reference image I^r (and its pose) and both relevant DVS poses for contrast calculation, as explained in Fig. H.3. Assuming the depth of the scene is known, the point $\mathbf{u}'$ in the reference image corresponding to the event location $(\mathbf{u}, t)$ in the DVS satisfies the following equation (in calibrated camera coordinates):

$$\mathbf{u}'(t) = \pi\Big(T_{RC}(t)\,\pi^{-1}\Big(\mathbf{u}, Z(t)\Big)\Big), \tag{H.12}$$

where $T_{RC}(t)$ is the transformation from the event camera frame at time t to the frame of the reference image, $Z(t)$ represents the scene structure (i.e., the depth of the map point corresponding to $\mathbf{u}$ with respect to the event camera), $\pi : \mathbb{R}^3 \to \mathbb{R}^2$, $(X, Y, Z) \mapsto (X/Z, Y/Z)$ is the canonical perspective projection, and π^{-1} is the inverse perspective projection. The transformation $T_{RC}(t_k)$ at the time of the current event depends on the current estimate of the event camera pose $\boldsymbol{\xi}_c \equiv \boldsymbol{\xi}(t_k)$ in (H.8); the poses $\boldsymbol{\xi}_i \equiv \boldsymbol{\xi}(t_i)$ and $\boldsymbol{\xi}_j \equiv \boldsymbol{\xi}(t_j)$ along the event camera trajectory $\boldsymbol{\xi}(t)$ enclosing the past timestamp $t_k - \Delta t$ are used to interpolate the pose $\boldsymbol{\xi}(t - \Delta t)$, which determines $T_{RC}(t_k - \Delta t)$. For simplicity, separate linear interpolations for position and rotation parameters (exponential coordinates) are used, although a Lie Group formulation with the $SE(3)$ exponential and logarithm maps (more computationally expensive) could be used.

Once the corresponding points of the event coordinates $(\mathbf{u}, t_k)$ and $(\mathbf{u}, t_k - \Delta t)$ have been computed, we use their intensity values on the reference image I^r to approximate the contrast:

$$\Delta \ln I \approx \ln I^r(\mathbf{u}'(t_k)) - \ln I^r(\mathbf{u}'(t_k - \Delta t)), \tag{H.13}$$

where t_k is the time of the current event and Δt is the time since the last event at the

same pixel. This approach is more accurate than linearizing $\Delta \ln I$. We assume that for a small pose change there is a relatively large number of events from different pixels. In this case the information contribution of a new event to an old pose will be negligible, and the new event will mostly contribute to the most recent pose.

Next, we linearize the measurement function in (H.11) around the expected state $\bar{s}_k = E_{p(s_k|o_{1:k-1})}[s_k]$, prior to incorporating the measurement correction:

$$\begin{aligned} M(o_k, \tilde{s}_k) &\approx M(o_k, \bar{\tilde{s}}_k) + \nabla_{\tilde{s}} M(o_k, \bar{\tilde{s}}_k) \cdot (\tilde{s}_k - \bar{\tilde{s}}_k) \\ &= \bar{M}_k + J_k \cdot \Delta\tilde{s}_k, \end{aligned} \tag{H.14}$$

where $\bar{M}_k$ and J_k are the predicted measurement and Jacobian at $\bar{s}_k$, respectively. Substituting in (H.11) we get:

$$p(o_k|s_k) = \pi_m \mathcal{N}(\bar{M}_k + J_k \cdot \Delta\tilde{s}_k; 0, \sigma_m^2) + (1 - \pi_m)\mathcal{U}. \tag{H.15}$$

We assume that the linearization is a good approximation to the original measurement function.

Finally, we may re-write the likelihood (H.15) in a more general and convenient form for deriving the filter equations, as a sum of exponential families for the state parameters s_k (see the Appendix):

$$p(o_k|s_k) = \sum_j h(s_k) \exp(\eta_{o,j} \cdot T(s_k) - A_{o,j}). \tag{H.16}$$

H.4.4 Posterior Approximation and Filter Equations

Our third contribution pertains to the approximation of the posterior distribution using a tractable distribution. For this, we consider variational inference theory [14], and choose a distribution in the exponential family as well as conjugate priors, minimizing the relative entropy error in representing the true posterior distribution with our approximate distribution, as we explain next.

Exponential families of distributions are useful in Bayesian estimation because they have *conjugate priors* [14]: if a given distribution is multiplied by a suitable prior, the resulting posterior has the same form as the prior. Such a prior is called a conjugate prior for the given distribution. The prior of a distribution in the exponential family is also in the exponential family, which clearly simplifies recursion. A mixture distribution like (H.16) does not, however, have a conjugate prior: the product of the likelihood and a prior from the exponential family is not in the family. Instead, the number of terms of the posterior doubles for each new measurement, making it unmanageable. Nevertheless, for tractability and flexibility, we choose as conjugate prior a distribution

in the exponential family and approximate the product, in the sense of the Kullback-Leibler (KL) divergence [71], by a distribution of the same form, as expressed by (H.6). This choice of prior is optimal if either the uniform or the normal terms of the likelihood dominates the mixture; we expect that small deviations from this still gives good approximations.

Letting the KL divergence (or relative entropy) from a distribution f to a distribution g be

$$D_{\mathrm{KL}}(f\|g) = \int f(x) \ln \frac{f(x)}{g(x)} dx, \tag{H.17}$$

which measures the information loss in representing distribution f by means of g, the posterior parameters η_k are calculated by minimization of the KL divergence from the distribution on the right hand side of (H.6) to the approximating posterior (left hand side of (H.6)):

$$\eta_k = \arg\min_{\eta} D_{\mathrm{KL}}\Big(C\, p(o_k|s_k) q(s_k;\eta_{k-1}) \| q(s_k;\eta)\Big).$$

It can be shown [14, p.505] that for g in the exponential family, the necessary optimality condition $\nabla_\eta D_{\mathrm{KL}}(f\|g) = 0$ gives the system of equations (in η)

$$E_{f(s)}[T(s)] = E_{g(s)}[T(s)], \tag{H.18}$$

i.e., the expected sufficient statistics must match. Additionally, the right hand side of (H.18) is $\nabla A \equiv \nabla_\eta A = E_{g(s)}[T(s)]$ since g is in the exponential family. In our case, $g \equiv q(s_k;\eta)$, $f \propto p(o_k|s_k) q(s_k;\eta_{k-1})$ and (H.18) can also be written in terms of the parameters of (H.16) [(H.26)-(H.27) in the Appendix], the log-normalizer A and its gradient:

$$\begin{aligned} 0 = \sum_j & \exp\Big(A(\eta_{o,j} + \eta_{k-1}) - A(\eta_{k-1}) - A_{o,j})\Big) \\ & \times \Big(\nabla A(\eta_{o,j} + \eta_{k-1}) - \nabla A(\eta)\Big). \end{aligned} \tag{H.19}$$

Equation (H.19) describes a system of equations that can be solved for η, yielding the update formula for η_k in terms of η_{k-1} and the current event o_k. For a multivariate Gaussian distribution over the DVS poses, explicit calculation of all update rules has the simple form of an Extended Kalman Filter (EKF) [63, 139] weighted by the inlier

Algorithm 3 Event-based pose tracking

Initialize state variables (DVS pose, contrast threshold, inlier ratio). Then, for each incoming event:
- propagate state covariance (zero-mean random diffusion)
- transfer the event to the map, compute the depth and evaluate the measurement function M function (H.14).
- compute K_k in (H.20), the inlier probability π_m, the weight w_k in (H.21), and the gain $w_k K_k$.
- update filter variables and covariance (e.g., (H.22)-(H.23)).

probability of that event:

$$K_k = P_k J_k^\top (J_k P_k J_k^\top + \sigma_m^2)^{-1} \tag{H.20}$$

$$w_k = \frac{\pi_m \mathcal{N}(\bar{M}_k; 0, \sigma_m^2)}{\pi_m \mathcal{N}(\bar{M}_k; 0, \sigma_m^2) + (1 - \pi_m)\mathcal{U}} \tag{H.21}$$

$$\boldsymbol{\xi}_{k+1} = \boldsymbol{\xi}_k + w_k K_k \bar{M}_k \tag{H.22}$$

$$P_{k+1} = (\mathbb{1} - w_k K_k J_k) P_k, \tag{H.23}$$

where $\mathbb{1}$ is the identity, $\bar{M}_k$ and J_k are given in (H.14), $\boldsymbol{\xi}$ are the 6-DOF coordinates (3 for translation and 3 for rotation) of the DVS pose, P is the pose covariance matrix, and $w_k K_k$ acts as the Kalman gain. A pseudocode of the approach is outlined in Algorithm 3.

The posterior approximation described in this section allows us to fuse the measurements and update the state-vector efficiently, without keeping multiple hypothesis in the style of particle filters, which would quickly become intractable due to the dimension of the state-vector.

H.5 Experimental Results

For our pose estimation algorithm to work, it requires an existing photometric map of the scene. As mentioned at the beginning of Section H.4, without loss of generality we describe the map in terms of depth maps with associated reference frames. These can be obtained from a previous mapping stage by means of an RGB-D camera or by classical dense reconstruction approaches using standard cameras (e.g., DTAM [106] or REMODE [114]), or even using a DVS (future research). In this work we use an Intel Realsense R200 RGB-D camera. We show experiments with both nearly planar scenes and scenes with large depth variations.

We evaluated the performance of our algorithm on several indoor and outdoor sequences. The datasets also contain fast motion with excitations in all six degrees of freedom (DOF).

First, we assessed the accuracy of our method against ground truth obtained by a motion-capture system. We placed the DVS in front of a scene consisting of rocks (Fig. H.4) at a mean scene depth of 60 cm and recorded eight sequences. Fig. H.4 shows the position and orientation errors (i.e., difference between the estimated ones and ground truth)[2] for one of the sequences, while Fig. H.9 shows the actual values of the estimated trajectory and ground truth over time. Fig. H.5 summarizes the errors

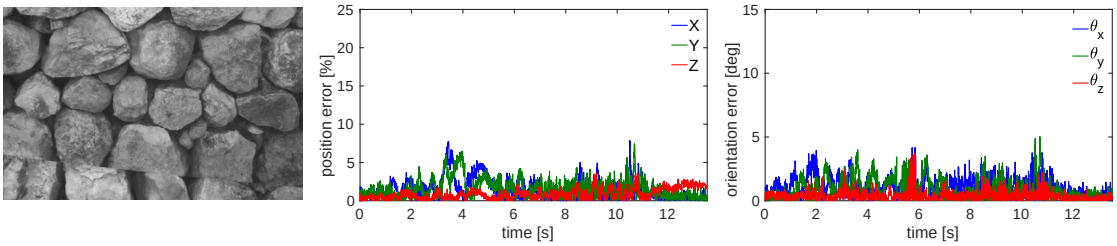


Figure H.4: Error plots in position (relative to a mean scene depth of 60 cm) and in orientation (in degrees) for one of the test sequences with ground truth provided by a motion capture system with sub-millimeter accuracy.

of the estimated trajectories for all sequences. The mean RMS errors in position and orientation are 1.63 cm and 2.21°, respectively, while the mean and standard deviations of the position and orientation errors are $\mu = 1.38$ cm, $\sigma = 0.84$ cm, and $\mu = 1.89°$, $\sigma = 1.15°$, respectively. Notice that the RMS position error corresponds to 2.71 % of the average scene depth, which is very good despite the poor spatial resolution of the DVS.

Next, we show the results of our algorithm in three outdoor sequences. In this case, ground truth is obtained via pose tracking with a standard camera running the SVO visual-odometry algorithm, which is available open source [50].[3]

To acquire accurate data for the evaluation, we rigidly mounted the DVS and the standard camera on a rig (see Figure H.6), and the same lens model was mounted on both sensors. The DVS has a spatial resolution of 128×128 pixels and operates asynchronously, in the microsecond scale. The standard camera is a global shutter MatrixVision Bluefox camera with a resolution of 752×480 pixels and a frame rate of up to 90 Hz. Both camera and DVS were calibrated intrinsically and extrinsically.

The three outdoor sequences (`ivy`, `graffiti`, and `building`) were recorded with the DVS-plus-camera rig viewing an ivy, a graffiti covered by some plants, and a building with people moving in front of it, respectively (see Fig. H.7, 1st column and accompanying video submission). The rig was moved by hand with increasing speed. All sequences exhibit significant translational and rotational motion. The error plots in position and orientation of all 6-DOFs are given in Fig. H.7. The reported error peaks in the `graffiti` and `building` sequences are due to a decrease of overlap between the DVS frustum and

[2]The rotation error is measured using the angle of their relative rotation (i.e., geodesic distance in $SO(3)$ [62]).

[3]SVO reports relative errors of 0.1%; hence it is justified to use its pose estimates as ground truth.

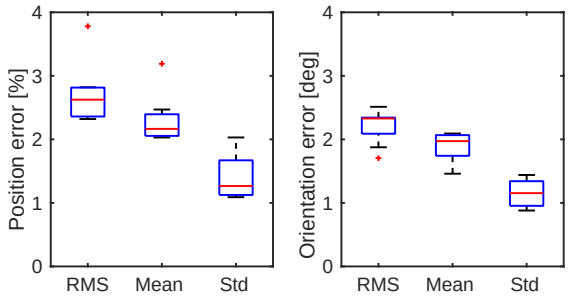


Figure H.5: Error in position (relative to a mean scene depth of 60 cm) and orientation (in degrees) of the trajectories recovered by our method for *all* `rocks` sequences (ground truth is given by a motion capture system). We provide box plots of the root-mean-square (RMS) errors, the mean errors and the standard deviation (Std) of the errors.

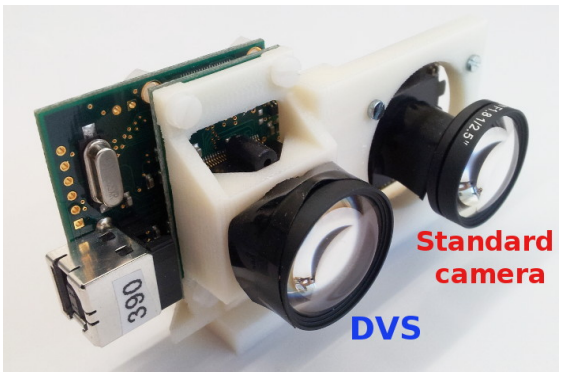


Figure H.6: A DVS and a standard camera mounted on a rig. The standard camera was only used for comparison.

the reference map, thus making pose estimation ambiguous for some motions (e.g., Y-translation vs. X-rotation).

Table H.1 summarizes the statistics of the pose tracking error for the three outdoor sequences. For the `ivy` dataset, the mean and standard deviation of the position error are 9.93 cm and 4.60 cm, which correspond to 3.97 % and 1.84 % of the average scene depth (2.5 m), respectively. The mean and standard deviation of the orientation error are 2.0° and 0.94°, respectively. For the `building` dataset, which presents the largest

Table H.1: Error measurements of three outdoor sequences. Translation errors are relative (i.e., scaled by the mean scene depth).

	Position error [%]			Orientation error [°]		
	RMS	μ	σ	RMS	μ	σ
`ivy`	4.37	3.97	1.84	2.21	2.00	0.94
`graffiti`	5.88	5.23	2.70	3.58	3.09	1.80
`building`	7.40	6.47	3.60	3.99	3.43	2.05

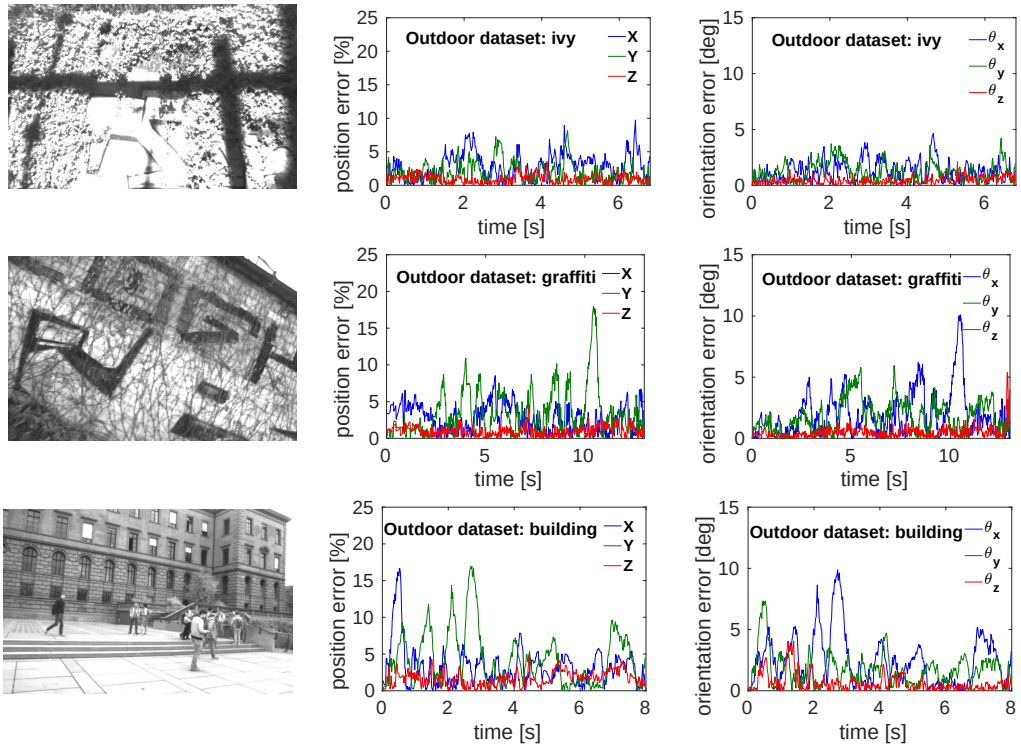


Figure H.7: Error plots in position (2nd column, relative to the mean scene depth) and in orientation (3rd column, in degrees) for three outdoor test sequences (1st column): `ivy`, `graffiti`, and `building`. The mean scene depths are 2.5 m, 3 m, and 30 m, respectively.

errors, the mean and standard deviation of the orientation error are 3.43° and 2.05°, respectively, while, in position error, the corresponding figures are 1.94 m and 1.08 m, that correspond to 6.47 % and 3.60 % of the average scene depth (30 m), respectively.

As reported by the small errors in Table H.1, overall our event-based algorithm is able to accurately track the pose of the DVS also outdoors. This shows that, in spite of the limited resolution of the DVS (128 × 128 pixels), the accuracy of the results provided by our event-based algorithm is comparable to that obtained by a standard camera processing 20× higher resolution images (752 × 480 pixels). This is made possible by the DVS temporal resolution being ten thousand times larger than the standard camera. We expect that the results provided by our approach would be even more accurate with the next generation of event-based sensors currently being developed [19, 75], which will have higher spatial resolution (640 × 480 pixels). Finally, observe that in the `building` sequence (Fig. H.7, bottom row), our method gracefully tracks the pose in spite of the considerable amount of events generated by moving objects (e.g., people) in the scene (see Figure H.8).

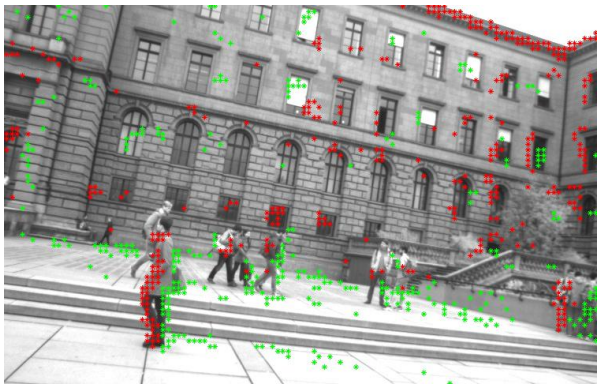


Figure H.8: The algorithm is able to track the DVS pose in spite of the considerable amount of events generated by moving objects (e.g., people) in the scene.

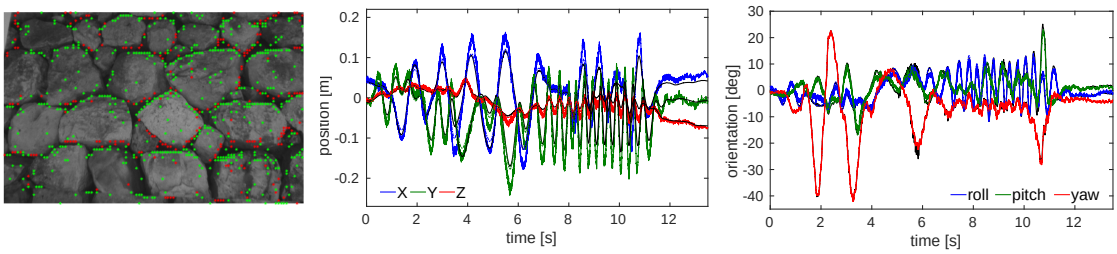


Figure H.9: Indoor experiment with 6-DOF motion. Left: Image of the standard camera overlaid with events (during mild motion). Events are displayed in red and green, according to polarity. Estimated position (center) and orientation (right) from our event-based algorithm (solid line), a frame-based method (dash-dot line) and ground truth (black line) from a motion capture system.

H.5.1 Tracking during High-Speed Motions

In addition to the error plots in Fig. H.4, we show in Fig. H.9 the actual values of the trajectories (position and orientation) acquired by the motion capture system (dashed line) and estimated by the event-based method (solid line) and frame-based method (dash-dot). Notice that they are all are almost indistinguishable relative to the amplitude of the motion excitation, which gives a better appreciation of the small errors reported in Figs. H.4 and H.5.

Figure H.10 shows a magnified version of the estimated trajectories during high-speed motions (occurring at $t \geq 7\,\text{s}$ in Fig. H.9). The frame-based method is able to track in the shaded region, up to $t \approx 8.66\,\text{s}$ (indicated by a vertical dashed line), at which point it loses tracking due to motion blur, while our event-based method continues to accurately estimate the pose.

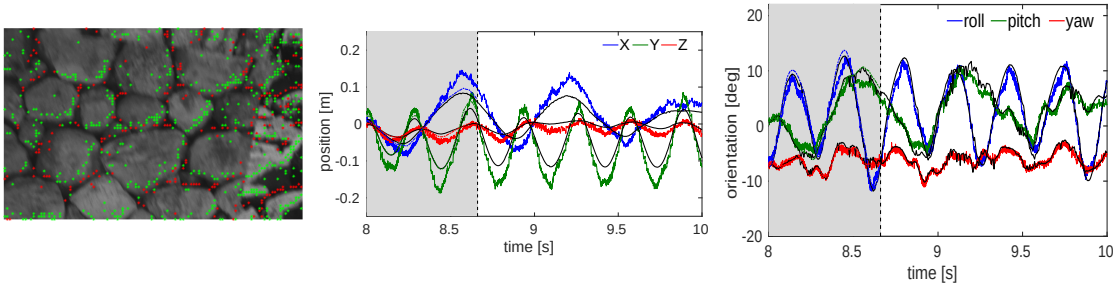


Figure H.10: Zoom of Fig. H.9. Left: Image of the standard camera overlaid with events (red and green points, according to polarity) during high-speed motion. Center and right: estimated trajectories. Due to the very high temporal resolution, our algorithm can still track the motion even when the images of the standard camera are sufficiently blurred so that the frame-based method (FB) failed. The event-based method (EB) provides pose updates even in high-speed motions, whereas the frame-based method loses track (it only provides pose updates in the region marked with the shaded area, then it fails).

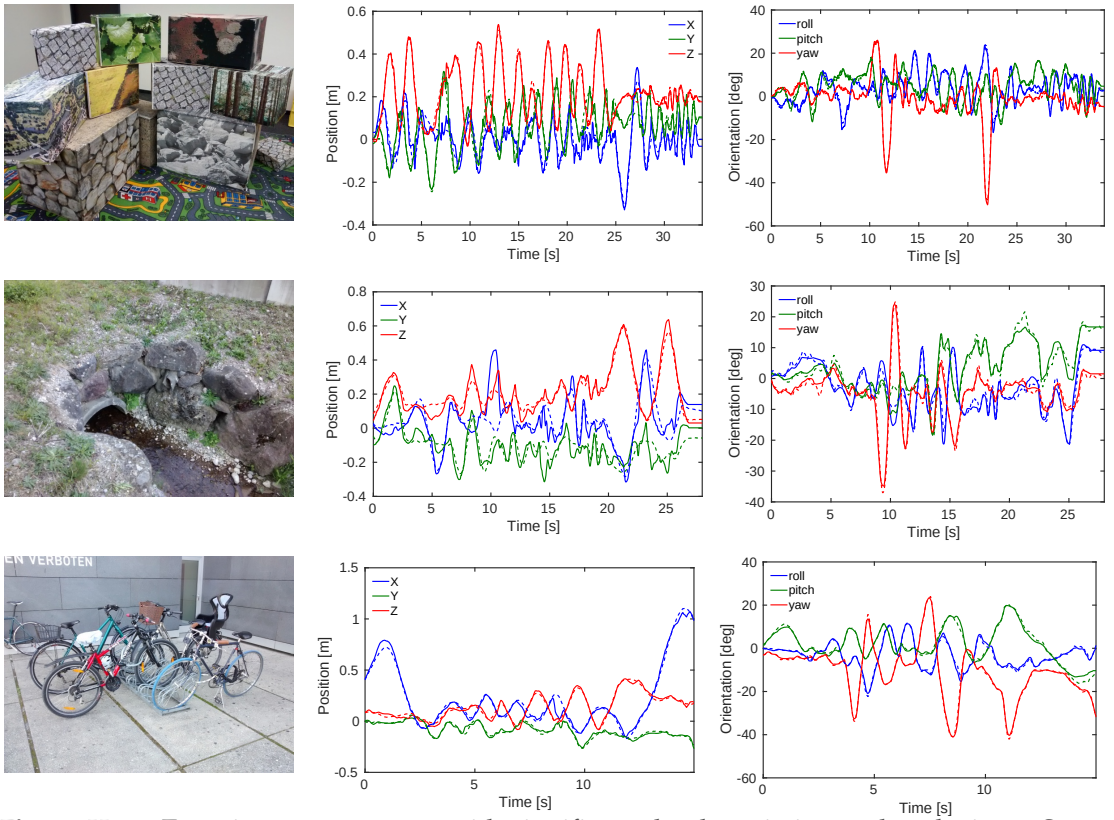


Figure H.11: Experiments on scenes with significant depth variation and occlusions. Scene impressions (1st column): `boxes`, `pipe`, and `bicycles`. Estimated position (2nd column, in meters) and orientation (3rd column, in degrees) from our event-based algorithm (solid line) compared with ground truth (dashed line). The mean scene depths are 1.8 m, 2.7 m, and 2.3 m, respectively.

H.5.2 Experiments with Large Depth Variation

In the following set of experiments, we also assessed the accuracy of our method on scenes with large depth variation and, therefore larger parallax than in previous

Table H.2: Error measurements of the sequences in Fig. H.11. Translation errors are relative (i.e., scaled by the mean scene depth).

	Position error [%]			Orientation error [°]		
	RMS	μ	σ	RMS	μ	σ
`boxes`	2.50	2.23	1.17	1.88	1.65	1.02
`pipe`	4.04	3.04	2.66	2.90	2.37	1.67
`bicycles`	2.14	1.724	1.27	1.46	1.19	0.84

experiments. We recorded seven sequences with ground truth from a motion-capture system of a scene consisting of a set of textured `boxes` (Fig. H.11, top row). We also recorded two outdoor sequences: `pipe` and `bicycles` (middle and bottom rows of Fig. H.11). The `pipe` sequence depicts a creek going through a pipe, surrounded by rocks and grass; the `bicycle` sequence depicts some parked bicycles next to a building; both outdoor scenes present some occlusions. All sequences exhibit significant translational and rotational motion.

Fig. H.12 summarizes the position and orientation error statistics of our algorithm on the `boxes` sequences (compared with ground truth from the motion-capture system). The position error is given relative to the mean scene depth, which is 1.9 m. As it is observed, the errors are very similar to those in Fig. H.5, meaning that our pose tracking method can handle arbitrary 3D scenes, i.e., not necessarily nearly planar.

Table H.2 reports the numerical values of the trajectory errors in both indoors and outdoor sequences. The row corresponding to the `boxes` sequences is the average of the errors in the seven indoor sequences (Fig. H.12). For the position error, the mean scene depths of the `pipe` and `bicycles` sequences are 2.7 m and 2.2 m, respectively. The mean RMS errors in position and orientation are in the range 2.5–4.0 % of the mean scene depth and 1.4–2.9°, respectively, which are in agreement with the values in Table H.1 for the scenes with mean depths smaller than 3 m. It is remarkable that the method is able to track despite some lack of texture (as in the `pipe` sequence, where there are only few strong edges), and in the presence of occlusions, which are more evident in the `bicycles` sequence.

H.5.3 Computational Effort

We measured the computational cost of our algorithm on a single core of an Intel(R) i7 processor at 2.60 GHz. The processing time per event is 32 µs, resulting in a processing event rate of 31.000 events per second. Depending on the texture of the scene and the speed of motion, the data rate produced by an event camera ranges from tens of thousands (moderate motion) to over a million events per second (high-speed motion). Our implementation is not optimal; many computations can be optimized, cached, and parallelized to increase the runtime performance.

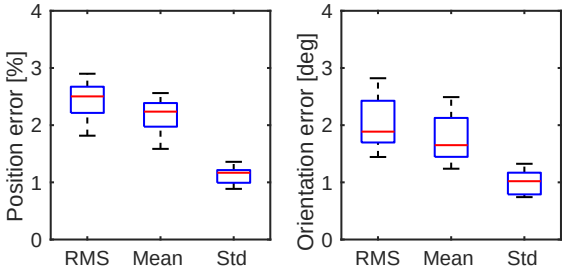


Figure H.12: Error in position (relative to a mean scene depth of 1.9 m) and orientation (in degrees) of the trajectories recovered by our method for *all* `boxes` sequences (ground truth is given by a motion-capture system). We provide box plots of the root-mean-square (RMS) errors, the mean errors and the standard deviation (Std) of the errors.

H.6 Conclusion

We have presented a novel, event-based probabilistic approach to track the pose of an arbitrarily moving event camera in 6-DOF in natural scenes. Our approach follows a Bayesian filtering methodology: the sensor model is given by a mixture-model likelihood that takes into account both the event-generation process and the presence of noise and outliers; the posterior distribution of the system state is approximated according to the relative-entropy criterion using distributions in the exponential family and conjugate priors. This yields a robust EKF-like filter that provides pose updates for every incoming event, at microsecond time resolution.

We have compared our method against ground truth provided by a motion capture system or a state-of-the-art frame-based pose-tracking pipeline. The experiments revealed that the proposed method accurately tracks the pose of the event-based camera, both in indoor and outdoor experiments in scenes with significant depth variation, and under motions with excitations in all 6-DOFs. In future, we plan to extend the proposed framework to a full event-based SLAM (Simultaneous Localization and Mapping) in 6-DOF.

Appendix: Rewriting the Likelihood Function

A distribution in the exponential family can be written as

$$p(x;\eta) = h(x)\exp\left(\eta \cdot T(x) - A(\eta)\right), \tag{H.24}$$

where η are the natural parameters, $T(x)$ are the sufficient statistics of x, $A(\eta)$ is the log-normalizer, and $h(x)$ is the base measure.

The likelihood (H.15) can be rewritten as:

$$\begin{aligned} p(o_k|s_k) =& \frac{1}{\sqrt{2\pi}} \exp(\ln(\pi_m) - \ln(\sigma_m) \\ & - \frac{1}{2}\left[J_k^i J_k^i \frac{\tilde{s}_k^i \tilde{s}_k^j}{\sigma_m^2} + 2\bar{M}_k J_k^i \frac{\tilde{s}_k^i}{\sigma_m^2} + \frac{\bar{M}_k^2}{\sigma_m^2} \right] \\ & + \exp(\ln((1-\pi_m)/(M_{\max} - M_{\min}))), \end{aligned} \tag{H.25}$$

where we use the Einstein summation convention for the indices of $J_k = (J_k^i)$ and $\tilde{s}_k = (\tilde{s}_k^i)$. Collecting the sufficient statistics into

$$T(s_k) = \left[\frac{\tilde{s}_k^i \tilde{s}_k^j}{\sigma_m^2}, \frac{\tilde{s}_k^i}{\sigma_m^2}, \frac{1}{\sigma_m^2}, \ln(\sigma_m), \ln(\pi_m), \ln(1-\pi_m) \right],$$

the likelihood can be conveniently rewritten as a sum of two exponential families (H.16), $j = 1, 2$, with $h(s) = 1$,

$$\eta_{o,1} = \left[-\frac{1}{2} J_k^i J_k^j, -\bar{M}_k J_k^i, -\frac{1}{2}\bar{M}_k^2, -1, 1, 0 \right] \tag{H.26}$$

$$\eta_{o,2} = \left[0_{ij}, 0_i, 0, 0, 1 \right] \tag{H.27}$$

$$A_{o,1} = \ln \sqrt{2\pi} \tag{H.28}$$

$$A_{o,2} = -\ln(M_{\max} - M_{\min}). \tag{H.29}$$

I Continuous-Time Visual-Inertial Trajectory Estimation

This chapter is a reprint of the article currently under review as:

E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza. "Continuous-Time Visual-Inertial Trajectory Estimation with Event Cameras". In: *IEEE Trans. Robot.* (2017). under review

A shorter version of this article was previously published as:

E. Mueggler, G. Gallego, and D. Scaramuzza. "Continuous-Time Trajectory Estimation for Event-based Vision Sensors". In: *Robotics: Science and Systems (RSS)*. 2015. DOI: 10.15607/RSS.2015.XI.036

Continuous-Time Visual-Inertial Trajectory Estimation with Event Cameras

Elias Mueggler, Guillermo Gallego, Henri Rebecq and Davide Scaramuzza

Abstract — Event cameras are bio-inspired vision sensors that output pixel-level brightness changes instead of standard intensity frames. They offer significant advantages over standard cameras, namely a very high dynamic range, no motion blur, and a latency in the order of microseconds. However, due to the fundamentally different structure of the sensor's output, new algorithms that exploit the high temporal resolution and the asynchronous nature of the sensor are required. Recent work has shown that a continuous-time representation of the trajectory to estimate can deal with the high temporal resolution and asynchronous nature of the event camera in a principled way. In this paper, we leverage a continuous-time representation to perform visual-inertial odometry with an event camera. This representation allows direct integration of the asynchronous events with micro-second accuracy and the inertial measurements at high frequency. The pose trajectory is approximated by a smooth curve in the space of rigid-body motions using cubic splines. This formulation significantly reduces the number of variables in trajectory estimation problems. We evaluate our method on real data from several scenes and compare the results against ground truth from a motion-capture system. We show superior performance of the proposed technique compared to non-batch event-based algorithms. We also show that both the map orientation and scale can be recovered accurately by fusing events and inertial data. To the best of our knowledge, this is the first work on visual-inertial fusion with event cameras using a continuous-time framework.

I.1 Introduction

Event cameras, such as the Dynamic Vision Sensor (DVS) [77], the DAVIS [19] or the ATIS [116], work very differently from a traditional camera. They have *independent* pixels that only send information (called "events") in presence of brightness changes in the scene at the time they occur. Thus, the output is not an intensity image but a stream of asynchronous events at *micro*-second resolution, where each event consists of its space-time coordinates and the *sign* of the brightness change (i.e., no intensity). Event cameras have numerous advantages over standard cameras: a latency in the order of microseconds, low power consumption, and a very high dynamic range (130 dB compared to 60 dB of standard cameras). Most importantly, since all the pixels are independent, such sensors do not suffer from motion blur.

However, because the output it produces—an event stream—is fundamentally different from video streams of standard cameras, new algorithms are required to deal with these data.

In this paper, we aim to use event cameras in combination with an Inertial Measurement Unit (IMU) for ego-motion estimation. This task, called Visual-Inertial Odometry (VIO), has important applications in various fields, such as mobile robotics and augmented/virtual reality (AR/VR) applications.

The approach provided by traditional visual-odometry frameworks, which estimate the camera pose at discrete times (naturally, the times the images are acquired), is no longer appropriate for event cameras, mainly due to two issues. First, a single event does not contain enough information to estimate the six degrees of freedom (DOF) pose of a calibrated camera. Second, it is not appropriate to simply consider several events for determining the pose using standard computer-vision techniques, such as PnP [68], because the events typically all have different timestamps, and so the resulting pose will not correspond to any particular time. Third, an event camera can easily transmit up to several million events per second, and, therefore, it can become intractable to estimate the pose of the event camera at the discrete times of all events due to the rapidly growing size of the state vector needed to represent all such poses.

To tackle the above-mentioned issues, we adopt a continuous-time framework [112]. Regarding the first two issues, an explicit continuous temporal model is a natural representation of the pose trajectory $\mathtt{T}(t)$ of the event camera since it unambiguously relates each event, occurring at time t_k, with its corresponding pose, $\mathtt{T}(t_k)$. To solve the third issue, the trajectory is described by a smooth parametric model, with significantly fewer parameters than events, hence achieving state space size reduction and computational efficiency. For example, to remove unnecessary states for the estimation of the trajectory of dynamic objects, [13] proposed to use cubic splines, reporting state-space size compression of 70–90 %. Cubic splines [52] or, in more general, Wavelets [5]

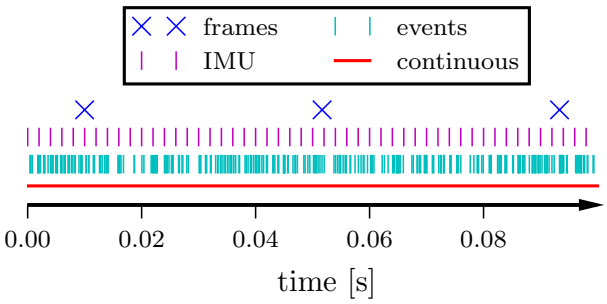


Figure I.1: While the frames and inertial measurements arrive at a constant frequency, events are transmitted asynchronously and at much higher frequency. We model the camera trajectory as continuous in time, which allows direct integration of all measurements using their precise timestamps.

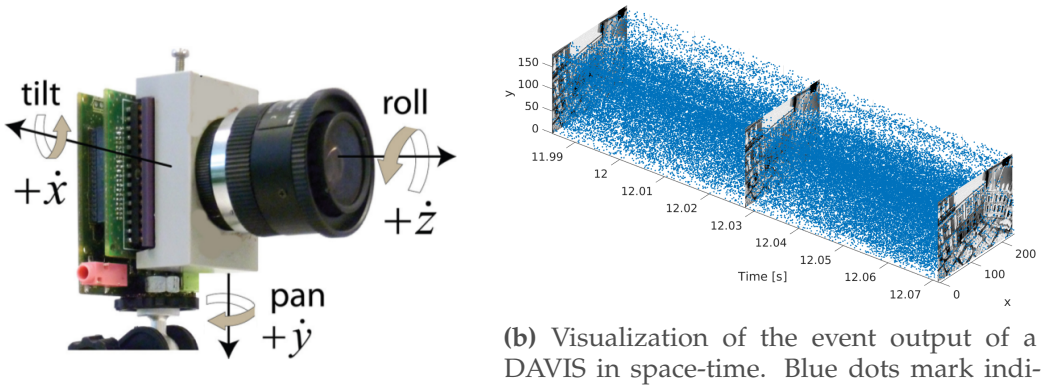
are common basis functions for continuous-time trajectories. The continuous-time framework was also motivated to allow data fusion of multiple sensors working at different rates and to enable increased temporal resolution [13]. This framework has been applied to camera-IMU fusion [52, 112], rolling-shutter cameras [112], actuated lidar [4], and RGB-D rolling shutter cameras [65].

Contribution

The use of a continuous-time framework for ego motion estimation with event cameras was first introduced in our previous conference paper [101]. In the present paper, we extend [101] in several ways:

- While in [101] we used the continuous time framework for trajectory estimation of an event camera only, here we tackle the problem of trajectory estimation by fusing an event camera with an inertial measurement unit. We show that the assimilation of inertial data allows us (i) to produce more accurate trajectories than with visual data alone and (ii) to estimate the absolute scale and orientation (alignment with respect to gravity).
- While [101] was limited to line-based maps, we extend the approach to work on natural scenes using point-based maps.
- We also show that our approach can be used to refine the poses estimated by an event-based visual-odometry method [124].
- We demonstrate the capabilities of the extended approach with new experiments, including natural scenes.

The paper is organized as follows: Section I.2 briefly introduces the principle of



(a) The DAVIS sensor from iniLabs (Figure adapted from [38]).

(b) Visualization of the event output of a DAVIS in space-time. Blue dots mark individual asynchronous events. The polarity of the events is not shown.

Figure I.2: The DAVIS camera and visualization of its output.

operation of event cameras, Section I.3 reviews previous work on ego-motion estimation with event cameras, Sections I.4 to I.6 present our method for continuous time trajectory optimization using visual-inertial event data fusion, Section I.7 presents the experiments carried out using the event camera to track two types of maps (point-based and line-based), Section I.8 discusses the results, and Section I.9 draws final conclusions.

I.2 Event Cameras

Standard cameras acquire frames (i.e., images) at fixed rates. On the other hand, event cameras such as the DAVIS [19] (Fig. I.2) have independent pixels that output brightness changes (called "events") asynchronously, at the time they occur. Specifically, if $L(\mathbf{u}, t) \doteq \ln I(\mathbf{u}, t)$ is the logarithmic brightness or intensity at pixel $\mathbf{u} = (x, y)^\top$ in the image plane, the DAVIS generates an event $e_k \doteq \langle x_k, y_k, t_k, p_k \rangle$ if the change in logarithmic brightness at pixel $\mathbf{u}_k = (x_k, y_k)^\top$ reaches a threshold C (typically 10-15% relative brightness change):

$$\Delta L(\mathbf{u}_k, t_k) \doteq L(\mathbf{u}_k, t_k) - L(\mathbf{u}_k, t_k - \Delta t) = p_k C, \tag{I.1}$$

where t_k is the timestamp of the event, Δt is the time since the previous event at the same pixel $\mathbf{u}_k$ and $p_k = \pm 1$ is the polarity of the event (the sign of the brightness change). Events are timestamped and transmitted asynchronously at the time they occur using a sophisticated digital circuitry.

Event cameras have the same optics as traditional perspective cameras, therefore, standard camera models (e.g., pinhole) still apply. In this work, we use the DAVIS 240C [19] that provides events and global-shutter images from the same physical pixels. In addition to the events and images, it contains a synchronized IMU. This

work solely uses the images for camera calibration, initialization and visualization purposes. The sensor's spatial resolution is 240×180 pixels and it is connected via USB. A visualization of the output of the DAVIS is shown in Fig. I.2b. An additional advantage of the DAVIS is its very high dynamic range of 130 dB (compared to 60 dB of high quality traditional image sensors).

I.3 Related Work: Ego-Motion Estimation with Event Cameras

A particle-filter approach for robot self-localization using the DVS was introduced in [145] and later extended to SLAM in [146]. However, the system was limited to planar motions and planar scenes parallel to the plane of motion, and scenes consisted of B&W line patterns.

In several works, conventional vision sensors have been attached to the event camera to simplify the ego-motion estimation problem. For example, [26] proposed an event-based probabilistic framework to update the relative pose of a DVS with respect to the last frame of an attached standard camera. The 3-D SLAM system in [144] relied on a frame-based RGB-D camera attached to the DVS to provide depth estimation, and thus build a voxel grid map that was used for pose tracking. The system in [70] used the intensity images from the DAVIS camera to detect features that were tracked using the events and were then fed into a 3-D visual odometry pipeline.

Robot localization in 6-DOF with respect to a map of B&W lines was demonstrated using a DVS, without additional sensing, during high-speed maneuvers of a quadrotor [102], where rotational speeds of up to $1{,}200\,^\circ/\mathrm{s}$ were measured. In natural scenes, [53] presented a probabilistic filter to track high-speed 6-DOF motions with respect to a map containing both depth and brightness information.

A system with two probabilistic filters operating in parallel was presented in [66] to estimate the rotational motion of a DVS and reconstruct the scene brightness in a high-resolution panorama. The system was extended in [67] using three filters that operated in parallel to estimate the 6-DOF pose of the event camera, and the depth and brightness of the scene.

More recently, [124] presented a geometric parallel-tracking-and-mapping approach for 6-DOF pose estimation and 3-D reconstruction with an event camera in natural scenes.

All previous methods operate in an event-by-event basis producing estimates of the event camera pose in a discrete, filter-like manner. This paper leverages a continuous-time representation of the trajectory of the event camera to couple the estimated poses in a tractable batch optimization that also allows to fuse event and inertial data.

I.4 Continuous-Time Trajectories

Traditional visual odometry and SLAM formulations use a discrete-time approach, i.e., the camera pose is calculated at the time the image was acquired. Recent works have shown that, for high-frequency data, a continuous-time formulation is preferable to keep the size of the optimization problem bounded [52, 112]. Temporal basis functions, such as B-splines, were proposed for camera-IMU calibration, where the frequencies of the two sensor modalities differ by an order of magnitude. While previous approaches use continuous-time representations mainly to reduce the computational complexity, in the case of an event-based sensor this representation is required to cope with the asynchronous nature of the events. Unlike a standard camera image, an event does not carry enough information to estimate the sensor pose by itself. A continuous-time trajectory can be evaluated at any time, in particular at each event's and inertial measurement's timestamp, yielding a well-defined pose for every event and well-defined derivatives. Thus, our method is not only computationally effective, but it is also necessary for a proper formulation.

I.4.1 Camera Pose Transformations

Following [112], we represent Euclidean space transformations between finite cameras [59, p. 157] by means of 4×4 matrices of the form

$$\mathtt{T}_{b,a} = \begin{bmatrix} \mathtt{R}_{b,a} & \mathbf{t}_a \\ \mathbf{0}^\top & 1 \end{bmatrix}, \tag{I.2}$$

where $\mathtt{R} \in SO(3)$ (the rotation group) and $\mathbf{t} \in \mathbb{R}^3$ are the rotational and translational components of the rigid-body motion, respectively. In homogeneous coordinates, a 3-D point in coordinate system a is mapped to a point in coordinate system b by the change of coordinates $\mathbf{X}_b \propto \mathtt{T}_{b,a}\mathbf{X}_a$. Transformations (I.2) form the special Euclidean group $SE(3)$ [88, p. 30], which has the structure of both a group and a differentiable manifold, i.e., a Lie group. A curve on $SE(3)$ physically represents the motion of a rigid body, e.g., the event camera. The tangent space of $SE(3)$ at the identity is $se(3)$, which has the structure of a Lie algebra. It corresponds to the space of *twists*, represented by 4×4 matrices of the form

$$\widehat{\boldsymbol{\xi}} = \begin{bmatrix} \widehat{\boldsymbol{\omega}} & \mathbf{v} \\ \mathbf{0}^\top & 0 \end{bmatrix}, \tag{I.3}$$

where $\mathbf{v} \in \mathbb{R}^3$ and $\widehat{\boldsymbol{\omega}}$ is the 3×3 skew-symmetric matrix representing the cross product: $\widehat{\boldsymbol{\omega}}\mathbf{b} = \boldsymbol{\omega} \times \mathbf{b}$, $\forall \boldsymbol{\omega}, \mathbf{b} \in \mathbb{R}^3$. Variables $\boldsymbol{\omega}$ and $\mathbf{v}$ physically represent the angular and linear velocity vectors of the moving sensor.

Based on the theory of Lie groups, the exponential map from $se(3)$ to $SE(3)$ can be

defined, which gives the Euclidean transformation associated to a twist, $\mathtt{T} = \exp(\widehat{\boldsymbol{\xi}})$. The inverse of the exponential map is the logarithmic map $\widehat{\boldsymbol{\xi}} = \ln(\mathtt{T})$. Moreover, every rigid-body motion $\mathtt{T} \in SE(3)$ can be represented in such an exponential parametrization, but the resulting twist may not be unique [88, p. 33]. However, to avoid this ambiguity, we adopt a local-chart approach (on the manifold $SE(3)$) by means of incremental rigid-body motions ($\mathtt{T} = \exp(\widehat{\boldsymbol{\xi}})$ with small matrix norm $\|\widehat{\boldsymbol{\xi}}\|$) given by the relative transformation between two nearby poses along the trajectory of the event camera (see (I.8)). In addition, this parametrization is free from singularities. Closed-form formulas for the exp and ln maps are given in [88].

The vee operator $[\cdot]^\vee$, inverse of the lift operator $\widehat{\cdot}$, maps a 3×3 skew-symmetric matrix to its corresponding vector,

$$\begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}^\vee = \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{I.4}$$

We can "linearly" interpolate between two poses at the extremes of an interval $[0, \Delta t] \ni t$ using formula

$$\mathtt{T}_w(t) = \mathtt{T}_{w,a} \exp\left(t \frac{1}{\Delta t} \ln\left(\mathtt{T}_{w,a}^{-1} \mathtt{T}_{w,b}\right)\right), \tag{I.5}$$

where w denotes the world coordinate system.

I.4.2 Cubic Spline Camera Trajectories in SE(3)

We use B-splines to represent continuous-time trajectories in $SE(3)$ for several reasons: they (*i*) are smooth (C^2 continuity in case of cubic splines), (*ii*) have local support, (*iii*) have analytical derivatives and integrals, (*iv*) interpolate the pose at any point in time, thus enabling data fusion from both asynchronous and synchronous sensors with different rates.

The continuous trajectory of the event camera is parametrized by control camera poses $\mathtt{T}_{w,i}$ at times t_i, $i \in \{0, \ldots, n\}$, where, according to the notation in (I.2), $\mathtt{T}_{w,i}$ is the transformation from the event-camera coordinate system at time t_i to a world coordinate system (w). Due to the locality of the cubic B-spline basis, the value of the spline curve at any time t only depends on four control poses: for $t \in [t_i, t_{i+1})$ such control poses occur at times $\{t_{i-1}, \ldots, t_{i+2}\}$. Following the cumulative cubic B-splines formulation [112], we use one absolute pose, $\mathtt{T}_{w,i-1}$, and three incremental poses, parameterized by twists (I.3) $\widehat{\boldsymbol{\xi}}_q \equiv \Omega_q$ (local approach on $SE(3)$). More specifically, the

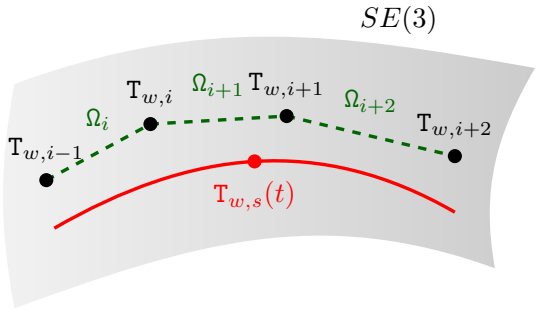


Figure I.3: Geometric interpretation of the cubic spline interpolation given by formula (I.6). The cumulative formulation uses one absolute control pose $\mathtt{T}_{w,i-1}$ and three incremental control poses $\Omega_i, \Omega_{i+1}, \Omega_{i+2}$ to compute the interpolated pose $\mathtt{T}_{w,s}$.

spline trajectory is given by

$$\mathtt{T}_{w,s}\Big(u(t)\Big) \doteq \mathtt{T}_{w,i-1} \prod_{j=1}^{3} \exp\left(\tilde{\mathbf{B}}_j\Big(u(t)\Big)\, \Omega_{i+j-1}\right), \tag{I.6}$$

where, for simplicity, we assume that the control poses are uniformly spaced in time [112], at $t_i = i\Delta t$, thus $u(t) = (t - t_i)/\Delta t \in [0, 1)$ is used in the cumulative basis functions for the B-splines,

$$\tilde{\mathbf{B}}(u) = \mathbf{C} \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}, \quad \mathbf{C} = \frac{1}{6} \begin{bmatrix} 6 & 0 & 0 & 0 \\ 5 & 3 & -3 & 1 \\ 1 & 3 & 3 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{I.7}$$

which are obtained from the matrix representation of the De Boor-Cox formula [121]. A graphical illustration of the difference between the standard and the cumulative basis functions for the B-splines is given in Fig. 2 of [112]. In (I.6), $\tilde{\mathbf{B}}_j$ is the j-th entry (0 based) of the cubic polynomial vector. The incremental pose from the coordinate system at t_{i-1} to the coordinate system at t_i in terms of world-referenced poses is encoded by the twist

$$\Omega_i = \ln(\mathtt{T}_{w,i-1}^{-1} \mathtt{T}_{w,i}). \tag{I.8}$$

Figure I.3 visualizes the evaluation of the pose $\mathtt{T}_{w,s}(t) \equiv \mathtt{T}_{w,s}(u(t))$ using one control pose $\mathtt{T}_{w,i-1}$ and three incremental poses Ω.

By the chain rule, the first and second temporal derivatives of the spline trajectory (I.6)

are, using Newton's dot notation for differentiation,

$$\dot{\mathtt{T}}_{w,s}(u) = \mathtt{T}_{w,i-1} \begin{pmatrix} \dot{\mathbf{A}}_0\mathbf{A}_1\mathbf{A}_2 \\ +\mathbf{A}_0\dot{\mathbf{A}}_1\mathbf{A}_2 \\ +\mathbf{A}_0\mathbf{A}_1\dot{\mathbf{A}}_2 \end{pmatrix}, \tag{I.9}$$

$$\ddot{\mathtt{T}}_{w,s}(u) = \mathtt{T}_{w,i-1} \begin{pmatrix} \ddot{\mathbf{A}}_0\mathbf{A}_1\mathbf{A}_2 + \mathbf{A}_0\ddot{\mathbf{A}}_1\mathbf{A}_2 \\ +\mathbf{A}_0\mathbf{A}_1\ddot{\mathbf{A}}_2 + 2\dot{\mathbf{A}}_0\dot{\mathbf{A}}_1\mathbf{A}_2 \\ +2\dot{\mathbf{A}}_0\mathbf{A}_1\dot{\mathbf{A}}_2 + 2\mathbf{A}_0\dot{\mathbf{A}}_1\dot{\mathbf{A}}_2 \end{pmatrix}, \tag{I.10}$$

respectively, where

$$\mathbf{A}_j \doteq \exp\left(\Omega_{i+j}\tilde{\mathbf{B}}(u)_{j+1}\right), \tag{I.11}$$

$$\dot{\mathbf{A}}_j = \mathbf{A}_j\Omega_{i+j}\dot{\tilde{\mathbf{B}}}(u)_{j+1}, \tag{I.12}$$

$$\ddot{\mathbf{A}}_j = \dot{\mathbf{A}}_j\Omega_{i+j}\dot{\tilde{\mathbf{B}}}(u)_{j+1} + \mathbf{A}_j\Omega_{i+j}\ddot{\tilde{\mathbf{B}}}(u)_{j+1}, \tag{I.13}$$

$$\dot{\tilde{\mathbf{B}}} = \frac{1}{\Delta t}\mathbf{C}\begin{bmatrix} 0 \\ 1 \\ 2u \\ 3u^2 \end{bmatrix}, \quad \ddot{\tilde{\mathbf{B}}} = \frac{1}{\Delta t^2}\mathbf{C}\begin{bmatrix} 0 \\ 0 \\ 2 \\ 6u \end{bmatrix}. \tag{I.14}$$

Analytical derivatives of $\mathtt{T}_{w,s}(u)$ with respect to the control poses are provided in the supplementary material of [65].

I.4.3 Visual and Inertial Predictions

A continuous trajectory model allows us to compute the velocity and acceleration of the event camera at any time. These quantities can be compared against IMU measurements and the resulting mismatch can be used to refine the modeled trajectory. Similarly to [112], the predictions of the IMU measurements, in angular velocity $\boldsymbol{\omega}$ and linear acceleration $\mathbf{a}$, are given by

$$\hat{\boldsymbol{\omega}}(u) \doteq \left(\mathtt{R}_{w,s}^{\top}(u) \cdot \dot{\mathtt{R}}_{w,s}(u)\right)^{\vee} + \mathbf{b}_{\omega}, \tag{I.15}$$

$$\hat{\mathbf{a}}(u) \doteq \left(\mathtt{R}_{w,s}^{\top}(u) \cdot (\ddot{\mathbf{s}}_w(u) + \mathbf{g}_w) + \mathbf{b}_a, \right. \tag{I.16}$$

where $\dot{\mathrm{R}}_{w,s}(u)$ is the upper-left 3×3 sub-matrix of $\dot{\mathrm{T}}_{w,s}\left(u\right)$, and $\ddot{\mathbf{s}}_w(u)$ is the upper-right 3×1 sub-matrix of $\ddot{\mathrm{T}}_{w,s}\left(u\right)$. $\mathbf{b}_\omega$ and $\mathbf{b}_a$ are the gyroscope and accelerometer biases, and $\mathbf{g}_w$ is the acceleration due to gravity in the world coordinate system. The vee operator $^\vee$ is defined in (I.4).

I.5 Map Representation

To focus on the event-camera trajectory estimation problem, we assume that the map of the scene is given and is time invariant. Specifically, the map $\mathcal{M}$ is either a set of 3-D points or 3-D line segments. We provide experiments using both geometric primitives.

In case of a map consisting of a set of points

$$\mathcal{M} = \{\mathbf{X}_i\}, \tag{I.17}$$

since events are caused by the apparent motion of edges, each 3-D point $\mathbf{X}_i$ represents a scene edge. Given a 3×4 projection matrix P modeling the perspective projection carried out by the event camera, the event coordinates are, in homogeneous coordinates, $\mathbf{u}_i \propto \mathrm{P}\mathbf{X}_i$.

In the case of lines, the map is

$$\mathcal{M} = \{\ell_j\}, \tag{I.18}$$

where each line segment ℓ_j is parametrized by its start and end points $\mathbf{X}_j^s, \mathbf{X}_j^e \in \mathbb{R}^3$. The lines of the map $\mathcal{M}$ can be projected to the image plane by projecting the endpoints of the segments. The homogeneous coordinates of the projected line through the j-th segment are

$$l_j \propto (\mathrm{P}\mathbf{X}_j^s) \times (\mathrm{P}\mathbf{X}_j^e). \tag{I.19}$$

I.6 Trajectory Optimization

In this section, we formulate the camera trajectory estimation problem from visual and inertial data in a probabilistic framework and derive the maximum likelihood solution (Section I.6.1). Then, to find a tractable solution, we reduce the dimensionality of the problem (Section I.6.2) by using the parametrized cubic spline trajectory representation introduced in Section I.4.2.

I.6.1 Probabilistic Approach

In general, the trajectory estimation problem over an interval $[0, T]$ can be cast in a probabilistic form [52], seeking an estimate of the joint posterior density $p(\mathbf{x}(t)|\mathcal{M}, \mathcal{Z})$ of the state $\mathbf{x}(t)$ (event camera trajectory) over the interval, given the map $\mathcal{M}$ and the set of all visual-inertial measurements, $\mathcal{Z} = \mathcal{E} \cup \mathcal{W} \cup \mathcal{A}$, which consists of: events $\mathcal{E} \doteq \{\mathbf{e}_k\}_{k=1}^{N}$ (where $\mathbf{e}_k = (x_k, y_k)^\top$ is the event location at time t_k), angular velocities $\mathcal{W} \doteq \{\boldsymbol{\omega}_j\}_{j=1}^{M}$ and linear accelerations $\mathcal{A} \doteq \{\mathbf{a}_j\}_{j=1}^{M}$. Using Bayes' rule, and assuming that the map is independent of the event camera trajectory, we may rewrite the posterior as

$$p(\mathbf{x}(t) \,|\, \mathcal{M}, \mathcal{E}, \mathcal{W}, \mathcal{A}) \propto p(\mathbf{x}(t))\, p(\mathcal{E}, \mathcal{W}, \mathcal{A} \,|\, \mathbf{x}(t), \mathcal{M}). \tag{I.20}$$

In the absence of prior belief for the state, $p(\mathbf{x}(t))$, the optimal trajectory is the one maximizing the likelihood $p(\mathcal{E}, \mathcal{W}, \mathcal{A} \,|\, \mathbf{x}(t), \mathcal{M})$. Assuming that the measurements $\mathcal{E}, \mathcal{W}, \mathcal{A}$ are independent of each other given the trajectory and the map, and using the fact that the inertial measurements do not depend on the map, the likelihood factorizes:

$$p(\mathcal{E}, \mathcal{W}, \mathcal{A}|\mathbf{x}(t), \mathcal{M}) = p(\mathcal{E}|\mathbf{x}(t), \mathcal{M})\; p(\mathcal{W}|\mathbf{x}(t))\; p(\mathcal{A} \,|\, \mathbf{x}(t)). \tag{I.21}$$

The first term in (I.21) comprises the visual measurements only. Under the assumption that the measurements $\mathbf{e}_k$ are independent of each other (given the trajectory and the map) and that the measurement error in the image coordinates of the events follows a zero-mean Gaussian distribution with variance σ_e^2, we have

$$\ln\Big(p(\mathcal{E} \,|\, \mathbf{x}(t), \mathcal{M})\Big) \tag{I.22}$$

$$= \ln\left(\prod_k p(\mathbf{e}_k|\mathbf{x}(t_k), \mathcal{M})\right) \tag{I.23}$$

$$= \ln\left(\prod_k K_1 \exp\left(-\frac{\|\mathbf{e}_k - \hat{\mathbf{e}}_k(\mathbf{x}(t_k), \mathcal{M})\|^2}{2\sigma_e^2}\right)\right) \tag{I.24}$$

$$= \tilde{K}_1 - \frac{1}{2}\sum_k \frac{1}{\sigma_e^2}\|\mathbf{e}_k - \hat{\mathbf{e}}_k(\mathbf{x}(t_k), \mathcal{M})\|^2 \tag{I.25}$$

where $K_1 \doteq 1/\sqrt{2\pi\sigma_e^2}$ and $\tilde{K}_1 \doteq \sum_k \ln K_1$ are constants (i.e., independent of the state $\mathbf{x}(t)$). Let us denote by $\hat{\mathbf{e}}_k(\mathbf{x}(t_k), \mathcal{M})$ the predicted value of the event location computed using the state $\mathbf{x}(t)$ and the map (I.18), $\mathcal{M}$. Such a prediction is a point on one of the projected 3-D primitives: in case of a map of points (I.17), $\hat{\mathbf{e}}$ is the projected point, and the norm in (I.25) is the standard reprojection error between two points; in case of a map of 3-D line segments (I.18), $\hat{\mathbf{e}}$ is a point on the projected line segment, and the norm in (I.25) is the Euclidean (orthogonal) distance from the observed point to

the corresponding line segment [101]. In both cases, (*i*) the prediction is computed using the event camera trajectory at the time of the event, t_i, and (*ii*) we assume the data association to be known, i.e., the correspondences between events and map primitives.[1] The likelihood (I.25) models only the error in the spatial domain, and not in the temporal domain since the latter is negligible: event timestamps have an accuracy in the order of a few dozen microseconds.

Following similar steps as those in (I.22)-(I.25) (independence and Gaussian error assumptions), the second and third terms in (I.21) lead to

$$\ln\Big(p(\mathcal{W}|\mathbf{x}(t))\Big) = \tilde{K}_2 - \frac{1}{2}\sum_j \frac{1}{\sigma_\omega^2}\|\boldsymbol{\omega}_j - \hat{\boldsymbol{\omega}}_j(\mathbf{x}(t_j))\|^2, \tag{I.26}$$

$$\ln\Big(p(\mathcal{A}|\mathbf{x}(t))\Big) = \tilde{K}_3 - \frac{1}{2}\sum_j \frac{1}{\sigma_a^2}\|\mathbf{a}_j - \hat{\mathbf{a}}_j(\mathbf{x}(t_j))\|^2, \tag{I.27}$$

where $\hat{\boldsymbol{\omega}}_j$ and $\hat{\mathbf{a}}_j$ are predictions of the angular velocity and linear acceleration of the event camera computed using the modeled trajectory $\mathbf{x}(t)$, such as those given by (I.15) and (I.16) in the case of a cubic spline trajectory.

Collecting terms (I.25)-(I.27), the maximization of the likelihood (I.21), or equivalently, its logarithm, leads to the minimization of the objective function

$$\begin{aligned} F \doteq & \frac{1}{N}\sum_{k=1}^{N}\frac{1}{\sigma_e^2}\|\mathbf{e}_k - \hat{\mathbf{e}}_k(\mathbf{x}(t_k), \mathcal{M})\|^2 \\ & + \frac{1}{M}\sum_{j=1}^{M}\frac{1}{\sigma_\omega^2}\|\boldsymbol{\omega}_j - \hat{\boldsymbol{\omega}}_j(\mathbf{x}(t_j))\|^2 + \frac{1}{M}\sum_{j=1}^{M}\frac{1}{\sigma_a^2}\|\mathbf{a}_j - \hat{\mathbf{a}}_j(\mathbf{x}(t_j))\|^2, \end{aligned} \tag{I.28}$$

where we omitted unnecessary constants. The first sum comprises the visual errors measured in the image plane and the last two sums comprise the inertial errors.

I.6.2 Constrained Optimization in Finite Dimensions

The objective function (I.28) is optimized with respect to the trajectory $\mathbf{x}(t)$ of the event camera, which in general is represented by an arbitrary curve in $SE(3)$, i.e., a "point" in an infinite-dimensional function space. However, because we represent the curve in terms of a finite set of known temporal basis functions (B-splines, formalized in (I.6)), the trajectory is parametrized by control poses $\mathtt{T}_{w,i}$ and, therefore, the optimization problem becomes finite dimensional. In particular, it is a non-linear least squares problem, for which standard numerical solvers such as Gauss-Newton or Levenberg-

[1] In practice, we solve the data association using event-based pose-tracking algorithms that we run as a preprocessing step. Note that these algorithms rely only on the events. Details are provided in the experiments of Section I.7.

Marquardt can be applied.

In addition to the control poses, we optimize with respect to model parameters $\boldsymbol{\theta} = (\mathbf{b}_\omega^\top, \mathbf{b}_a^\top, s, \mathbf{o}^\top)^\top$, consisting of the IMU biases $\mathbf{b}_\omega$ and $\mathbf{b}_a$, and the map scale s and orientation with respect to the gravity direction $\mathbf{o}$. The map orientation $\mathbf{o}$ is composed of roll and pitch angles, $\mathbf{o} = (\alpha, \beta)^\top$. Maps obtained by monocular systems, such as [124, 67], lack information about absolute map scale and orientation, so it is necessary to estimate them in such cases. We estimate the trajectory and additional model parameters by minimizing the objective function (I.28),

$$\{\mathtt{T}_{w,i}^*, \boldsymbol{\theta}^*\} = \arg\min_{\mathtt{T},\boldsymbol{\theta}} F. \tag{I.29}$$

This optimization problem is solved in an iterative way using the Ceres solver [2], an efficient numerical implementation for non-linear least squares problems.

One final remark: the inertial predictions are computed as described in (I.15) and (I.16), using $\mathtt{T}_{w,s}$ and its derivatives, whereas the visual predictions require the computation of $\mathtt{T}_{w,s}^{-1}$. More specifically, for each event e_k, triggered at time t_k in the interval $[t_i, t_{i+1})$, we compute its pose $\mathtt{T}_{w,s}(u_k)$ using (I.6), where $u_k = (t_k - t_i)/\Delta t$. We then project the map point or line segment into the current image plane using projection matrices $\mathtt{P}(t_k) \propto \mathtt{K}(\mathtt{I}|\mathbf{0})\mathtt{T}_{w,s}^{-1}(t_k)$, $\mathtt{K}$ being the time-invariant intrinsic parameter matrix of the event camera (after radial distortion compensation), and compute the distance between the event location $\mathbf{e}_k$ and the corresponding point $\hat{\mathbf{e}}_k$ in the projected primitive. To take into account the map scale s and orientation $\mathbf{o}$, we right-multiply $\mathtt{P}(t_k)$ by a similarity transformation with scale s and rotation $\mathtt{R}(\mathbf{o})$ before projecting the map primitives.

I.7 Experiments

We evaluate our method on several datasets using the two different map representations in Section I.5: lines-based maps and point-based maps. These two representations allow us to evaluate the effect of two different visual error terms, which are the line-to-point distance and point-to-point reprojection errors presented in Section I.6.1. In both cases, we quantify the trajectory accuracy using the ground truth of a motion-capture system.[2] We use the same hand-eye calibration method as described in [103]. Having a monocular setup, the absolute scale is not observable from visual observations alone. However, we are able to estimate the absolute scale since the fused IMU measurements grant scale observability. The following two sections describe the experiments with line-based maps and point-based maps, respectively. In these experiments, we used $\sigma_e = 0.1\,\text{pixel}$, $\sigma_\omega = 0.03\,\text{rad/s}$, and $\sigma_a = 0.1\,\text{m}^2/\text{s}$. We chose the values for the standard deviations of the inertial measurements to be around ten times higher than

[2] We use a NaturalPoint OptiTrack system with 14 motion-capture cameras spanning a volume of $100\,\text{m}^3$. The system reported a calibration accuracy of 0.105 mm and provides measurements at 200 Hz.

those measured at rest.

I.7.1 Trajectory Estimation in Line-based Maps

These experiments are similar to the ones presented in [101]. Here, however, we use the DAVIS instead of the DVS, which provides the following advantages. First, it has a higher spatial resolution of 240×180 pixels (instead of 128×128 pixels). Second, it provides inertial measurements (at 1 kHz) that are time-synchronized with the events. Third, it also outputs global-shutter intensity images (at 24 Hz) that we use for initialization, visualization, and a more accurate intrinsic camera calibration than that achieved using events.

Tracking Method

This method tracks a set of lines in a given metric map. Event-based line tracking is done using [102], which also provides data association between events and lines. This data association is used in the optimization of I.29. Events that are not close to any line (such as the keyboard and mouse in Fig. I.4a) are not considered to be part of the map and, therefore, are ignored in the optimization. Pose estimation is then done using the Gold Standard PnP algorithm [59, p.181] on the intersection points of the lines. Fig. I.4 shows tracking of two different shapes.

Experiment

We moved the DAVIS sensor by hand in a motion-capture system above a square pattern, as shown in Fig. I.5a. The corresponding error plots in position and orientation are shown in Fig. I.5b: position error is measured using the Euclidean distance, whereas orientation error is measured using the geodesic distance in $SO(3)$ (the angle of the relative rotation between the true rotation and the estimated one) [62]. The excitation in each degree of freedom and the corresponding six error plots are shown in Fig. I.9. The error statistics are summarized in Table I.1.

We compare three algorithms against ground truth from a motion-capture system: (*i*) the event-based tracking algorithm in [102] (in cyan), (*ii*) the proposed spline-based optimization without IMU measurements (blue color), and (*iii*) the proposed spline-based optimization (with IMU measurements, in red color). As it can be seen in the figures and in Table I.1, the proposed spline-based optimization ("`Spline (ev.+IMU)`" label) is more accurate than the event-based tracking algorithms: the mean, standard deviation, and maximum errors in both position and orientation are the smallest among all methods (last row of Table I.1). The mean position error is 0.5 % of the average scene depth and the mean orientation error is 0.37°. The errors are up to five times smaller

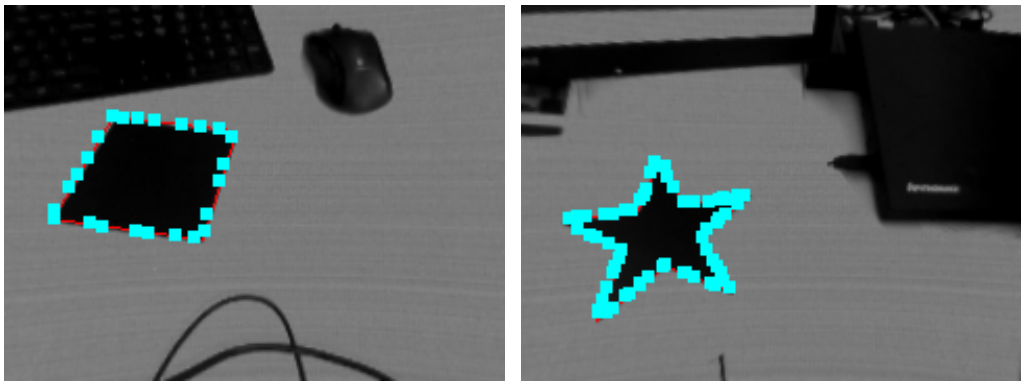
compared to the event-based tracking method (cf. rows 1 and 3 in Table I.1). Hence, the proposed method is very accurate. The benefit of including the inertial measurements in the optimization is also reported: the vision-only spline-based optimization method is better than the event-based tracking algorithm [102] (by approximately a factor of 1.5). However, when inertial measurements are included in the optimization the errors are reduced by a factor of 4 approximately (by comparing rows 2 and 3 of Table I.1). Therefore, there is a significant gain in accuracy ($\times 4$ in this experiment) due to the fusion of inertial measurements and the event data to estimate the sensor's trajectory.

For this experiment, we placed control poses at an interval of 0.1 s. This leads to ratios of about 5000 events and 100 inertial measurements per control pose. We initialize the control poses by fitting a spline trajectory through the initial tracker poses.

Scale Estimation In further experiments, we also estimate the absolute scale s of the map as an additional parameter. As we know the map size precisely, we report the relative error. The square shape has a side length of 10 cm. For these experiments, we set the initial length to 0.1 cm, 1 cm, 1 m, and 10 m (two orders of magnitude in both directions). The optimization converged to virtually the same minimum and the relative error was below 7 % for all cases. This error is in the same ballpark as the magnitude error of the IMU, which we measured to be about 5 % (10.30 $\mathrm{m/s^2}$ instead of 9.81 $\mathrm{m/s^2}$ when the sensor is at rest).

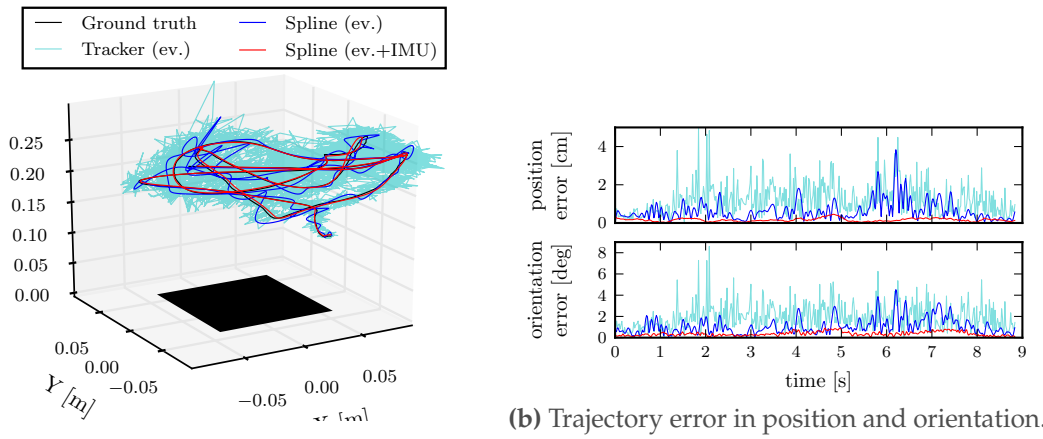
I.7.2 Trajectory Estimation in Point-based Maps

The following experiments show that the proposed continuous-time trajectory estimation also works on natural scenes, i.e., without requiring strong artificial gradients to generate the events. For this, we used three sequences from the Event-Camera Dataset [103], which we refer to as *desk*, *boxes* and *dynamic* (see Figs. I.6a, I.7a, and I.8a). The *desk* scene features a desktop with some office objects (books, a screen, a keyboard, etc.); the *boxes* scene features some boxes on a carpet, and the *dynamic* scene consists of a desk with objects and a person moving them. All datasets were recorded hand-held and contain data from the DAVIS (events, frames, and inertial measurements) as well as ground-truth pose measurements from a motion-capture system (at 200 Hz). We processed the data with EVO [124], an event-based visual-odometry algorithm, that we describe below, which also returns a point-based map of the scene. Then, we used the events and the point-based map of EVO for batch trajectory optimization in the continuous-time framework, showing that we achieve higher accuracy and a smoother trajectory.



(a) Square shape. **(b)** Star shape.

Figure I.4: Screenshots of the line-based tracking algorithm. The lines and the events used for its representation are in red and cyan, respectively. The image is only used for initialization and visualization.



(a) Estimated trajectories and 3D map. **(b)** Trajectory error in position and orientation. Legend as in Fig. I.5a.

Figure I.5: Results on Line-based Tracking and Pose Estimation.

Table I.1: Results on Line-based Tracking and Pose Estimation. Position and orientation errors.

	Position error (abs. [cm] and rel. [%])						Orientation error [°]		
	μ	%	σ	%	max	%	μ	σ	max
Tracker (ev.) [102]	1.11	3.43	0.75	2.31	7.96	24.56	1.87	1.13	9.61
Spline (ev.)	0.64	1.98	0.51	1.57	3.85	11.87	1.08	0.75	4.55
Spline (ev.+IMU)	0.16	0.50	0.08	0.26	0.46	1.41	0.37	0.19	0.93

Relative errors are given with respect to the mean scene depth.

Tracking Method

EVO [124] returns both a map and a set of 6-DOF discrete, asynchronous poses of the event camera. In a post-processing step, we extracted the correspondences between

the events and the map points that are required to optimize (I.29). We project the map points onto the image plane for each pose of EVO and establish a correspondence if a projected map point and an event are present in the same pixel. Events that cannot be associated with a map point are treated as noise and are therefore ignored in the optimization. Figs. I.6b, I.7b, and I.8b show typical point-based maps produced by EVO, projected onto the image plane and colored according to depth with respect to the camera. The same plots also show all the observed events, colored in gray. Notice that the projected map is aligned with the observed events, as expected from an accurate tracking algorithm. The corresponding scenes are shown in Figs. I.6a, I.7a, and I.8a, respectively.

Experiments

Figs. I.6–I.8 and Tables I.2–I.4 summarize the results obtained on the three datasets. Fig. I.6c, I.7c, and I.8c show the 3D maps and the event camera trajectories. Figs. I.6d, I.7d and I.8d show the position and orientation errors obtained by comparing the estimated trajectories against motion-capture ground truth. Error statistics are provided in Tables I.2, I.3 and I.4. In additional plots in the Appendix (Figs. I.9 to I.12), we show the individual degrees of freedom and their errors, respectively.

We compare four methods against ground truth from the motion-capture system: (*i*) event-based pose tracking using EVO (in cyan color in the figures), (*ii*) spline-based trajectory optimization without IMU measurements (in blue color), (*iii*) spline-based trajectory optimization (events and inertial measurements, in red color), and (*iv*) spline-based trajectory and absolute scale optimization (in magenta). The output trajectory of each of the first three methods was aligned with respect to ground truth using a 3D similarity transformation (rotation, translation, and uniform scaling); thus, the absolute scale is externally provided. Although a Euclidean alignment suffices (rotation and translation, without scaling) for the spline-based approach with events and IMU, we also used a similarity alignment for a fair comparison with respect to other methods. The fourth method has the same optimized trajectory as the third one, but the alignment with respect to the ground truth trajectory is Euclidean (6-DOF): the absolute scale is recovered from the inertial measurements. As it can be seen in Tables I.2, I.3, and I.4, the spline-based approach without inertial measurements consistently achieves smaller errors than EVO (cf. rows 1 and 2 of the tables). Using also the inertial measurements further improves the results (cf. rows 2 and 3 of the tables). When using the estimated absolute map scale, the results are comparable to those where the scale was provided by ground-truth alignment with a similarity transform, even though a low-cost IMU was used (cf. rows 3 and 4 of the tables). In such a case, the mean position error is less than 1.05 % of the average scene depth, and the mean orientation error is less than 1.03°. The standard deviations of the errors are also very small: less than 0.43 % and less than 0.57°, respectively, in all datasets. The results are remarkably accurate. The gain

in accuracy due to incorporating the inertial measurements in the optimization (with respect to the visual-only approach) is less than a factor of two, which is not as large as in the case of line-based maps (a factor of four) because EVO [124] already provides very good results compared with the line-based tracker of [102]. Nevertheless, the gain is still significant, making the event-inertial optimization consistently outperforming the event-only one.

We placed the knots (the timestamps of the control poses $\mathtt{T}_{w,i}$) at a time interval of 0.2 s, 0.15 s, and 0.15 s for the *desk*, *boxes*, and *dynamic* datasets, respectively. This leads to a ratio of about 10^4 events and 150–200 inertial measurements per control pose. We initialize the control poses by fitting a spline through the initial tracker poses.

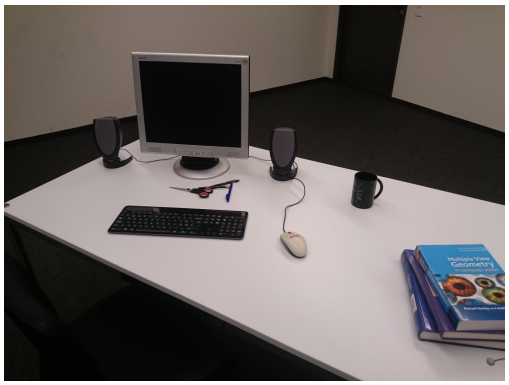
Absolute Map Scale and Gravity Alignment

In the above experiments with IMU, we also estimated the absolute scale s and orientation $\mathbf{o}$ of the map as additional parameters. Since EVO is monocular, it cannot estimate the absolute scale. However, by fusing the inertial data with EVO, it is possible to recover the absolute scale and to align the map with gravity. We found that the absolute scale deviated from the true value by 4.1 %, 6.5 %, and 2.8 % for the *desk*, *boxes*, and *dynamic* datasets, respectively. For the alignment with gravity, we found that the estimated gravity direction deviated from the true value by 3.83°, 20.18°, and 3.34° for the *desk*, *boxes*, and *dynamic* datasets, respectively. The high alignment error for the *boxes* dataset is likely due to the dominant translational motion of the camera.

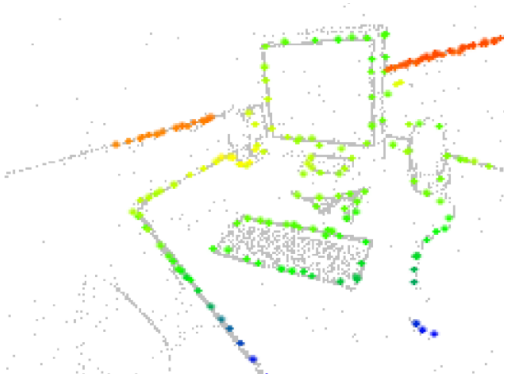
I.7.3 Computational Cost

Table I.5 reports the runtime for the least-squares optimization of (I.29) using the Ceres library [2] and the number of iterations taken to converge to a tolerance of 10^{-3} in the change of the objective function value. The table also provides an overview of the experiments (dataset duration, number of events and inertial measurements, and number of control poses used). The experiments were conducted on a laptop with an Intel Core i7-3720QM CPU at 2.60 GHz.

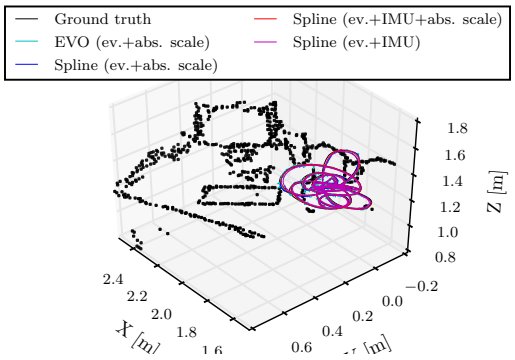
The optimization process typically converges within a few iterations (ten or less). Depending on the number of iterations, our approach is around three to ten times slower than real-time. Most of the computation time (around 80 %) is devoted to the evaluation of Jacobians, which is done using automatic differentiation. The optimization could be made real time by adding more computational power (such as a GPU), by following a sliding-window approach, or by using analytical derivatives and approximations, such as using the same pose derivative for several measurements that are close in time.



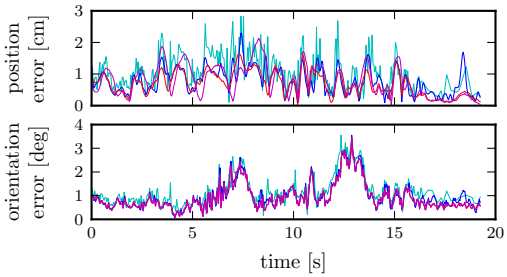
(a) Scene *desk*



(b) Event-based tracking. Events (gray) and reprojected map (colored using depth).



(c) Estimated trajectories and 3D map.



(d) Trajectory error in position and orientation. Legend as in Fig. I.6c.

Figure I.6: Results for *desk* dataset.

Table I.2: Results for *desk* dataset.

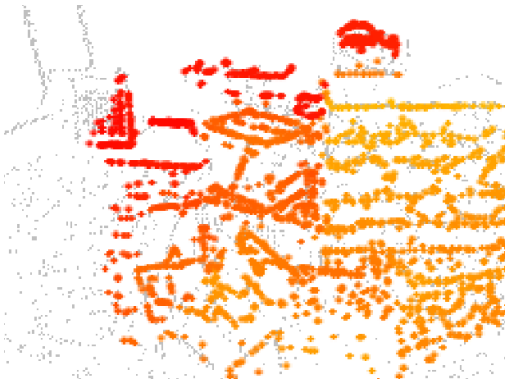
	Position error (abs. [cm] and rel. [%])						Orientation error [°]		
	μ	%	σ	%	max	%	μ	σ	max
EVO (ev.+abs. scale)	1.08	0.54	0.53	0.27	4.64	2.33	1.31	0.68	3.55
Spline (ev.+abs. scale)	0.78	0.39	0.40	0.20	2.30	1.16	0.98	0.58	3.56
Spline (ev.+IMU+abs. scale)	0.69	0.35	0.37	0.18	1.65	0.83	0.93	0.56	3.45
Spline (ev.+IMU)	0.79	0.39	0.48	0.24	2.13	1.07	0.93	0.56	3.45

Relative errors are given with respect to the mean scene depth.

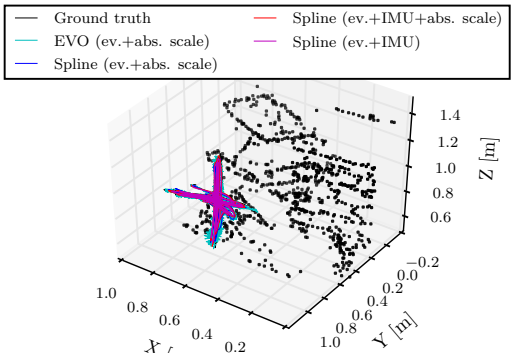
I.8 Discussion

Event cameras provide visual measurements asynchronously and at very high rate. Traditional formulations, which describe the camera trajectory using poses at discrete timestamps, are not appropriate to deal with such almost continuous data streams because of the difficulty in establishing correspondences between the discrete sets of events and poses, and because the preservation of the temporal information of the events would require a very large number of poses (one per event). The continuous-time

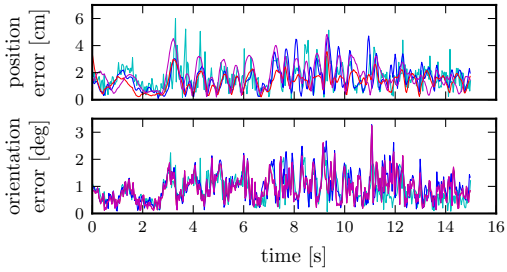
(a) Scene *boxes*



(b) Event-based tracking. Events (gray) and reprojected map (colored using depth).



(c) Estimated trajectories and 3D map.



(d) Trajectory error in position and orientation. Legend as in (c).

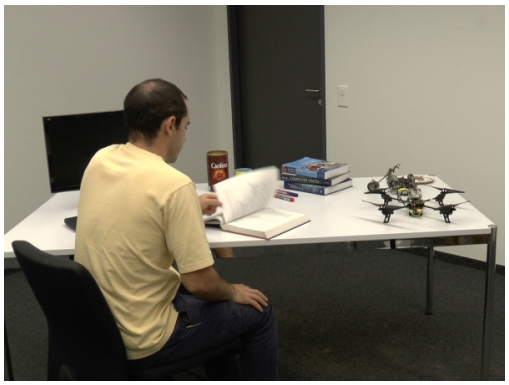
Figure I.7: Results for *boxes* dataset.

Table I.3: Results for *boxes* dataset.

	Position error (abs. [cm] and rel. [%])						Orientation error [°]		
	μ	%	σ	%	max	%	μ	σ	max
EVO (ev.+abs. scale)	1.66	0.62	0.88	0.33	6.83	2.56	0.99	0.50	2.77
Spline (ev.+abs. scale)	1.58	0.59	0.89	0.34	4.72	1.77	0.99	0.54	3.28
Spline (ev.+IMU+abs. scale)	1.34	0.50	0.62	0.23	3.59	1.35	0.91	0.49	3.24
Spline (ev.+IMU)	1.77	0.66	0.93	0.35	4.87	1.82	0.91	0.49	3.24

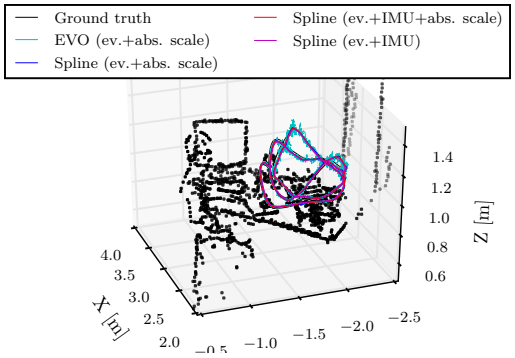
Relative errors are given with respect to the mean scene depth.

framework is a convenient representation of the camera trajectory since it has many desirable properties, among them: (*i*) it solves the issue of establishing correspondences between events and poses (since the pose at the time of the event is well-defined), and (*ii*) it is a natural framework for data fusion: it deals with the asynchronous nature of the events as well the synchronous samples from the IMU. As demonstrated in the experiments, such event-inertial data fusion allows significantly increasing the accuracy of the estimated camera motion over event-only–based approaches (e.g., by a factor of

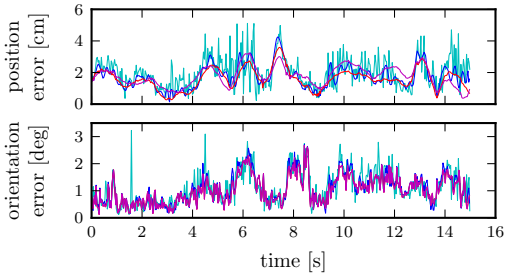


(a) Scene *dynamic*

(b) Event-based tracking. Events (gray) and reprojected map (colored using depth).



(c) Estimated trajectories and 3D map.



(d) Trajectory error in position and orientation. Legend as in (c).

Figure I.8: Results for *dynamic* dataset.

Table I.4: Results for *dynamic* dataset.

	Position error (abs. [cm] and rel. [%])						Orientation error [°]		
	μ	%	σ	%	max	%	μ	σ	max
EVO (ev.+abs. scale)	1.94	1.39	0.94	0.68	7.06	5.07	1.08	0.58	3.66
Spline (ev.+abs. scale)	1.74	1.25	0.74	0.53	6.40	4.59	1.08	0.56	3.42
Spline (ev.+IMU+abs. scale)	1.60	1.14	0.64	0.46	3.62	2.59	1.02	0.51	3.43
Spline (ev.+IMU)	1.78	1.27	0.62	0.45	3.23	2.32	1.02	0.51	3.43

Relative errors are given with respect to the mean scene depth.

four).

The proposed parametric B-spline model makes the trajectory optimization computationally feasible since it has local basis functions (i.e., sparse Hessian matrix), analytical derivatives (i.e., fast to compute), and it is a compact representation: few parameters (control poses) suffice to assimilate several hundred thousand events and inertial measurements while providing a smooth trajectory. Additionally, batch optimization (si-

Table I.5: Dataset Statistics and Computational Cost of the Optimization (I.29)

	Dataset Statistics				Computational Cost			
					Events-only		Events + IMU	
Experiment	# Events	# IMU	# Control Poses	Duration [s]	Time [s]	Iterations	Time [s]	Iterations
line-based	450,416	8,842	92	8.8	28.4	2	48.0	3
desk	883,449	19,317	99	19.3	110.7	8	181.0	7
boxes	2,064,028	14,977	103	15.0	82.0	1	407.7	8
dynamic	879,143	14,976	103	15.0	35.1	1	216.9	10

multaneous estimation of poses by exploiting their coupling), as opposed to filter-based approaches, is the preferred strategy to achieve maximum accuracy, at the expense of introducing some processing latency [49]. In this sense, our method demonstrated its usefulness to refine trajectories from state-of-the-art event-based pose trackers such as EVO, with or without inertial measurements. The current implementation runs off-line, as a post-processing stage, but the method can be adapted for on-line processing in a temporal sliding-window manner; the local support of the B-spline basis functions enables such type of local temporal processing.

Another reason for adopting the continuous-time trajectory framework is that it is agnostic to the map representation. We showed that the proposed method is flexible, capable of estimating accurate camera trajectories in scenes with line-based maps as well as point-based maps. In fact, the probabilistic (maximum likelihood) justification of the optimization approach gracefully unifies both formulations, lines and points, in the same objective function in a principled way. In this manner, we extended the method in [101] and broadened its applicability to different types of maps (i.e., scenes). The probabilistic formulation also allows a straightforward generalization to other error distributions besides the normal one. More specifically, the results of the proposed method on line-based and point-based maps show similar remarkable accuracy, with mean position error of less than 1 % of the average scene depth, and mean orientation error of less than 1°. The absolute scale and gravity direction are recovered in both types of maps, with an accuracy of approximately 5 %, which matches the accuracy of the IMU accelerometer; thus, the proposed method takes full advantage of the accuracy of the available sensor.

Using cumulative B-splines in SE(3) sets a prior on the shape of the trajectory. While smooth rigid-body motions are well-approximated with such basis functions, they are not suitable to fit discontinuities (such as bumps or crashes). In this work, we use fixed temporal spacing of the control poses, which is not optimal when the motion speed changes abruptly within a dataset. Choosing the optimal number of control poses and their temporal spacing is beyond the scope of this paper and is left for future work.

I.9 Conclusion

In this paper, we presented a visual-inertial odometry method for event cameras using a continuous-time framework. This approach can deal with the asynchronous nature of the events and the high frequency of the inertial measurements in a principled way while providing a compact and smooth representation of the trajectory using a parametric model. The pose trajectory is approximated by a smooth curve in the space of rigid-body motions using cubic splines. The approximated trajectory is then optimized according to a geometrically meaningful error measure in the image plane and a direct inclusion of the inertial measurements, which have probabilistic justifications. We tested our method on real data from two recent algorithms: a simple line-based tracker and an event-based visual-odometry algorithm that works on natural scenes. In all experiments, our method outperformed previous algorithms when comparing to ground truth, with a remarkable accuracy: mean position error of less than 1 % of the average scene depth, and mean orientation error of less than 1°.

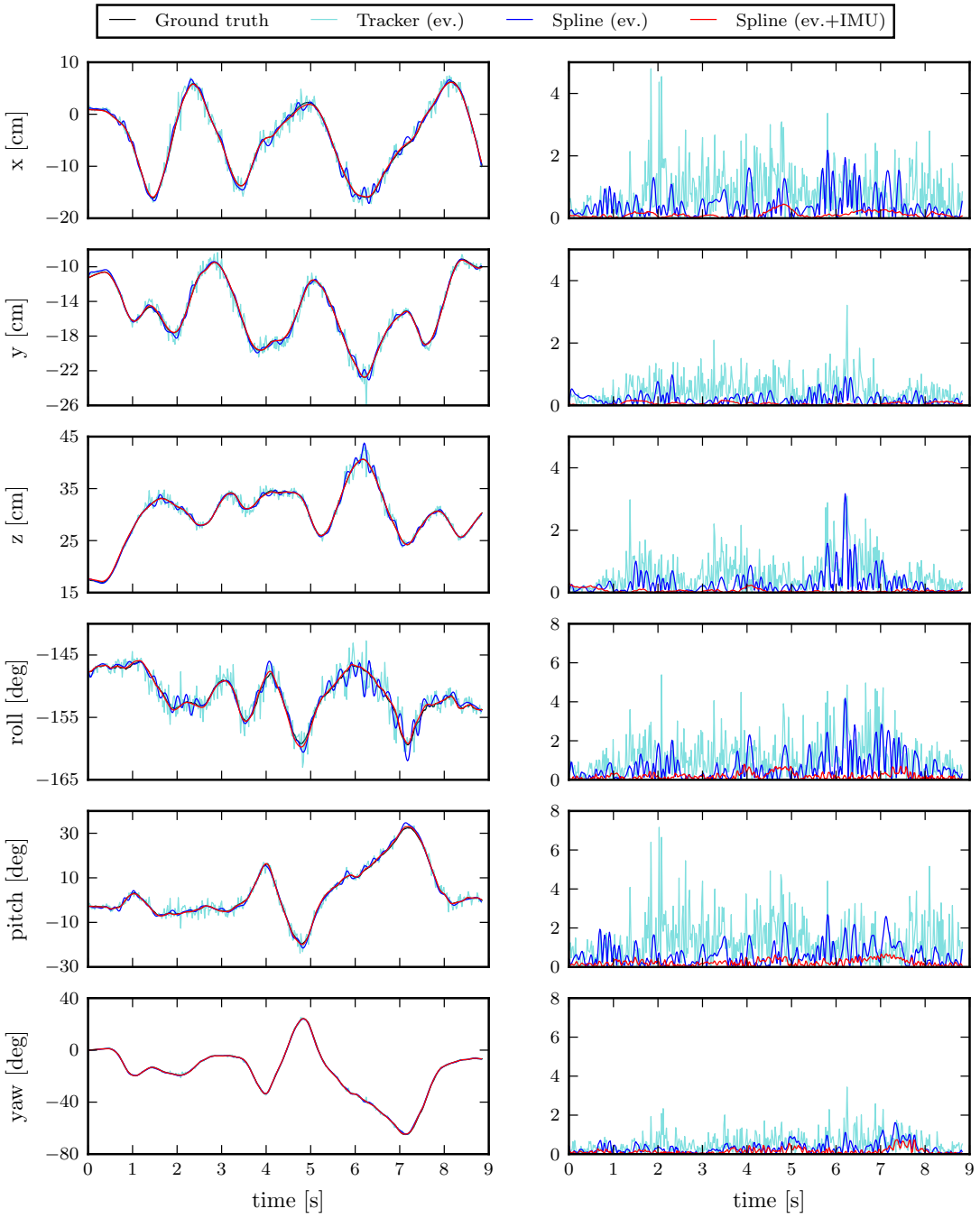


Figure I.9: *line-based* dataset. Plots of the 6-DOF (left column) and error (right column) of the estimated trajectories in Fig. I.5a.

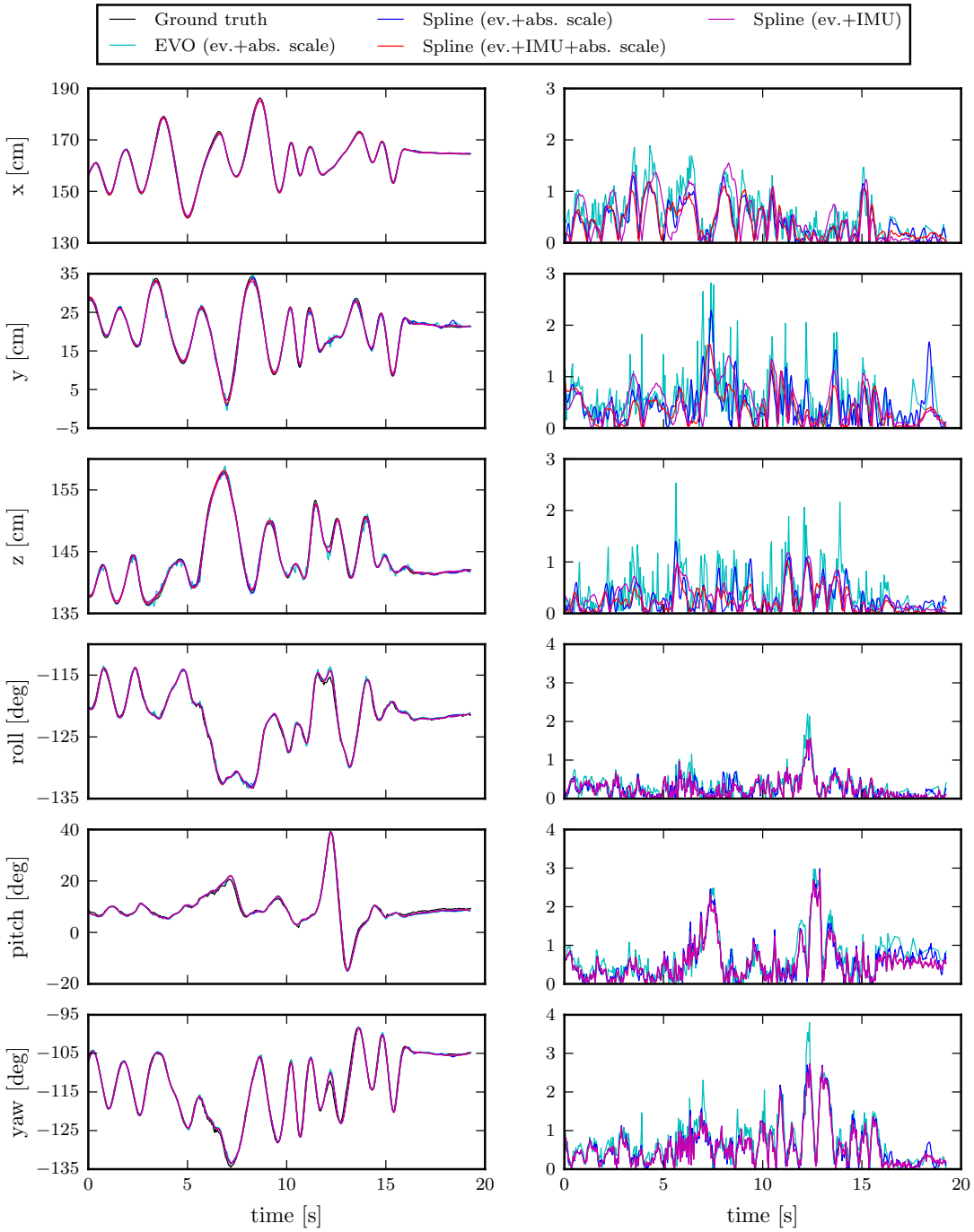


Figure I.10: *desk* dataset. Plots of the 6-DOF (left column) and error (right column) of the estimated trajectories in Fig. I.6c.

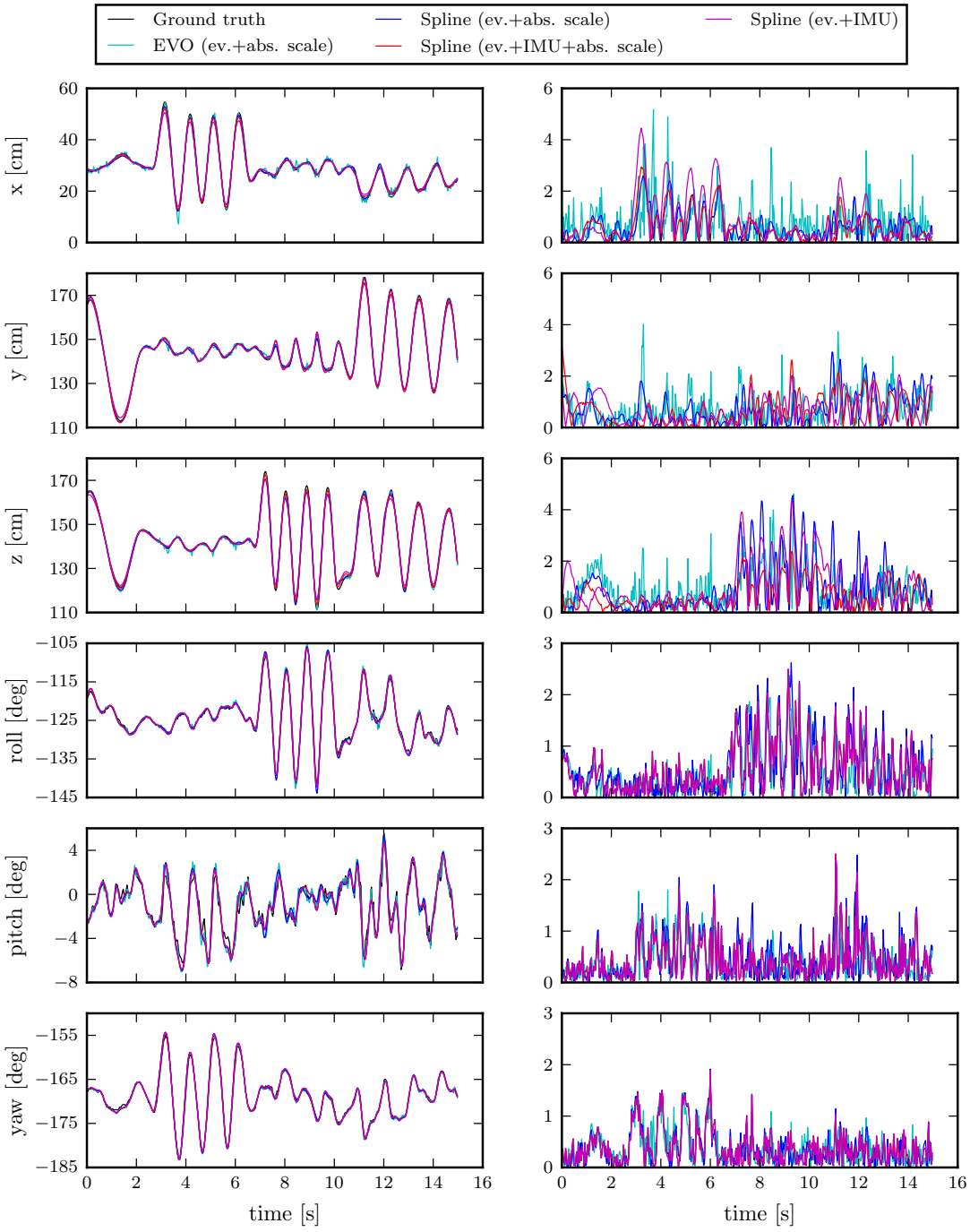


Figure I.11: *boxes* dataset. Plots of the 6-DOF (left column) and error (right column) of the estimated trajectories in Fig. I.7c.

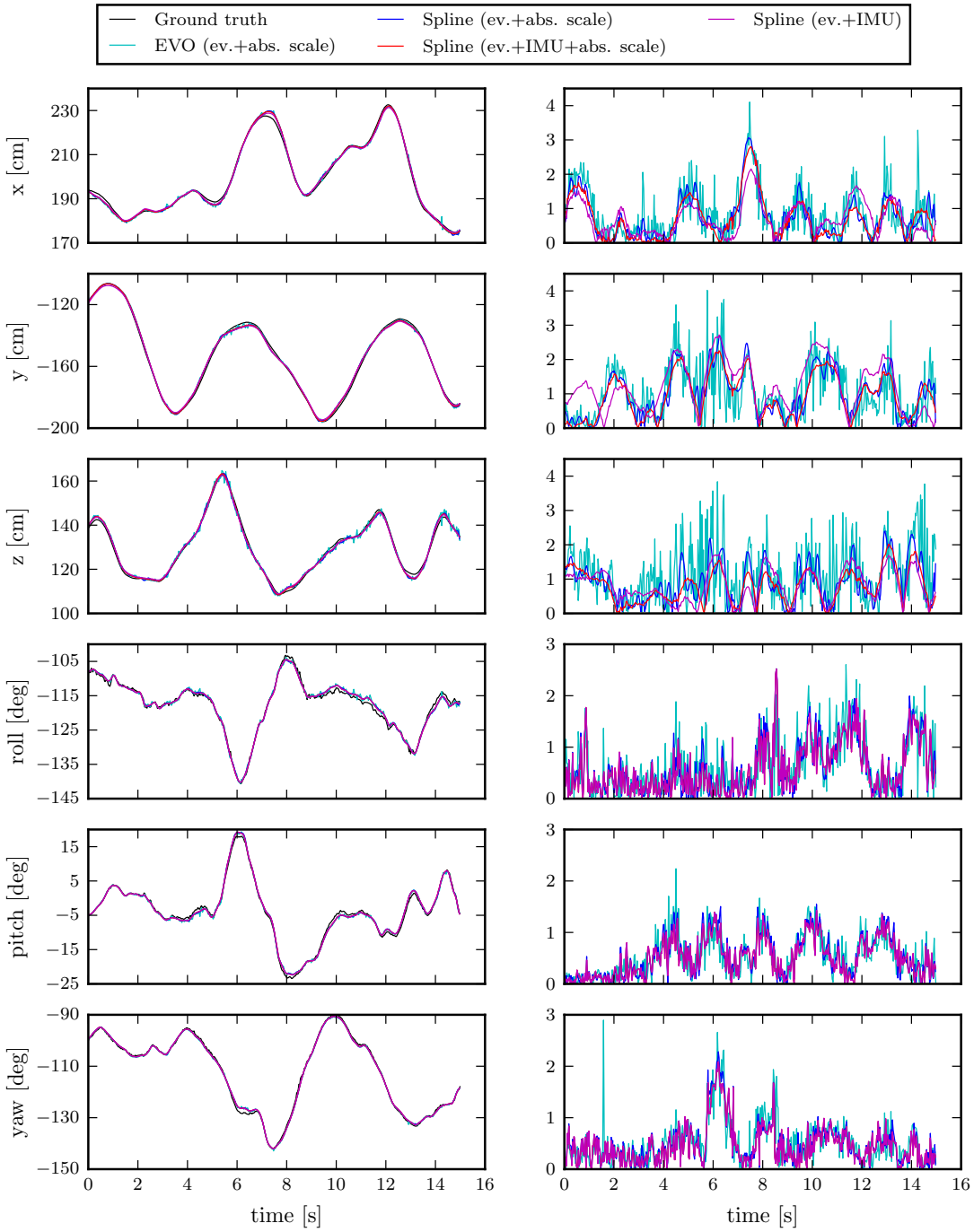


Figure I.12: *dynamic* dataset. Plots of the 6-DOF (left column) and error (right column) of the estimated trajectories in Fig. I.8c

J Slot-Car Racing

T. Delbruck, M. Pfeiffer, R. Juston, G. Orchard, E. Müggler, A. Linares-Barranco, and M. W. Tilden. "Human vs. Computer Slot Car Racing using an Event and Frame-Based DAVIS Vision Sensor". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. Lisbon, Portugal, May 2015, pp. 2409–2412. DOI: 10.1109/ISCAS.2015.7169170

Human vs. Computer Slot Car Racing using an Event and Frame-Based DAVIS Vision Sensor

Tobi Delbruck, Michael Pfeiffer, Raphaël Juston, Garrick Orchard, Elias Müggler, Alejandro Linares-Barranco and Mark W. Tilden

Abstract — This paper describes an open-source implementation of an event-based dynamic and active pixel vision sensor (DAVIS) for racing human vs. computer on a slot car track. The DAVIS is mounted in "eye-of-god" view. The DAVIS image frames are only used for setup and are subsequently turned off because they are not needed. The dynamic vision sensor (DVS) events are then used to track both the human and computer controlled cars. The precise control of throttle and braking afforded by the low latency of the sensor output enables consistent out-performance of human drivers at a laptop CPU load of <3 % and update rate of 666 Hz. The sparse output of the DVS event stream results in a data rate that is about 1000 times smaller than from a frame-based camera with the same resolution and update rate. The scaled average lap speed of the 1/64 scale cars is about 450 km/h which is twice as fast as the fastest Formula 1 lap speed. A feedback-controller mode allows competitive racing by slowing the computer controlled car when it is ahead of the human. In tests of human vs. computer racing the computer still won more than 80 % of the races.

J.1 Introduction

The DAVIS is a neuromorphic camera that outputs static image frames concurrently with dynamic vision sensor (DVS) temporal contrast events [77, 19]. DVS address-

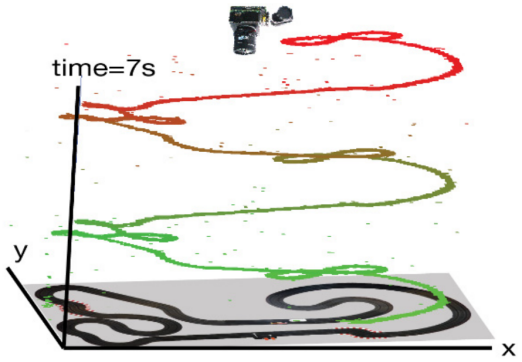


Figure J.1: A track layout with the stream of DVS events (dots) caused by a moving car shown as dots in 3D space-time. The average event rate caused by a moving car is about 5k events/sec.

events (AEs) asynchronously signal changes of log intensity. The AE timestamp (in microseconds) codes the time of the events. Pixels with DVS output are neuromorphic abstractions of retinal ganglion cells in biological retinas. Their sub-ms latency, sparse output, and kHz pixel bandwidth has led to applications requiring high speed object tracking with short-latency feedback, e.g. [16, 34]. In this work, we use the DVS outputs to track slot cars and control one of the cars to race competitively against human drivers. In this application, the DAVIS static frames were useful for setting up the sensor and adjusting focusing, but subsequently the frames were not needed and were turned off.

J.2 Hardware and Software Setup

Fig. J.1 shows a racetrack together with sample DVS data produced by a single car driving around the track. The 240x180 pixel DAVIS was mounted in "eye of god" view over the table using a wide angle 2.6 mm lens with a horizontal field of view of 81° to cover the slot car track. As the cars go around the track, they create DVS events, which are shown superimposed as 3D space-time events over the photo of the track. These events are used as described later to track the cars and to control the computer car throttle and braking. The events captured from the DAVIS are transmitted to the host PC over a USB interface. Individual events are time-stamped with 1 µs resolution.

Slot cars contain a DC motor, a pin to guide the car along the slot in the track, two brass brushes that pick up power from the metal rails of the track, and a magnet that helps hold the car onto the steel track rails. Power to the car is normally regulated by a simple throttle controller consisting of a wire-wound resistor. Racers attempt to go as fast as possible around the track without flying off it. We used a HO-scale (1/64) system from AFX Racing (www.afxracing.com) with car chassis type SRT. The cars are 7 cm in length, and the track (Fig. J.2) had 15 turns in a length of 805 cm, or 900 pixels on the sensor image. The fastest lap times are about 4.1 s. The slot car speed of

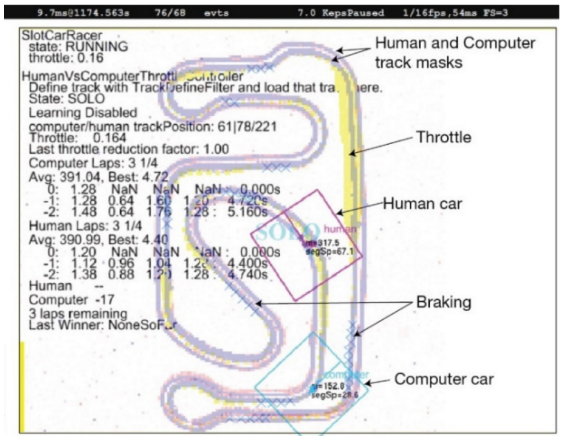


Figure J.2: Display of the slot car track with state information.

200 cm/s scaled to Formula 1 size is 450 km/h; for reference, the fastest average lap speed achieved in a Formula 1 race was 248 km/h on a track with 11 turns (Monza 2003). Cars can accelerate to full speed in about 300 ms and decelerate by friction in about the same time. The electronic motor braking (described later) slows the car even faster, allowing more aggressive driving.

Implementation The software implementation of the slot car racer is open-sourced in the jAER project [1] in the package *ch.unizh.ini.jaer.projects.virtualslotcar*.[2] The main slot car racer class is *SlotCarRacer*. The throttle controller described in this paper is the class *HumanVsComputerThrottleController*.

J.2.1 Car Tracking, Track Model, and Track Masking

The operator sees a view of the track and other information superimposed on the DAVIS sensor output as shown in Fig. J.2. Different parts of this display are labeled and are referred to below.

Tracking (computed by the class *CarTracker*) uses a model of the car consisting of a rectangle that is constrained to move along the track model, which is a list of track vertices in sensor pixel coordinates. A *CarCluster* is a software object based on [34] that has a 2D pixel position and a velocity along the track in track vertices per second. The track model is obtained in a semi-automated way in *TrackDefineFilter* by driving each car around alone, collecting a 2D event histogram, and then extracting a list of vertices spaced by minimum distances, starting from the peak of the histogram. This list is then

[1] jAER Open Source Project (2007): http://jaerproject.org

[2] http://sourceforge.net/p/jaer/code/HEAD/tree/jAER/trunk/src/ch/unizh/ini/jaer/projects/virtualslotcar (r5522)

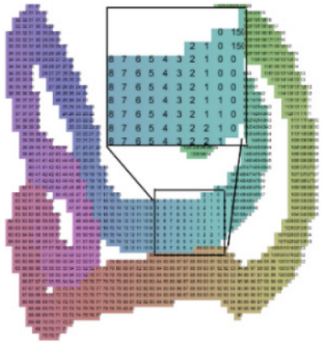


Figure J.3: Mapping from each pixel location to the nearest track vertex speeds up lookup of the nearest track vertex (number at each pixel location).

updated manually using a GUI to drag, add, and delete vertices.

Fig. J.4 illustrates the car tracking update. Each DVS event input to *CarTracker* is first used to update the current car position according to the car velocity along the track, using the last update time and the current event time. This update implements the model inertia. Then the event is checked if it is near the location of the tracked car (shaded rectangle in Fig. J.4 and boundaries of boxes surrounding cars in Fig. J.2). If so, the car model is updated by either advancing or retarding the car position along the track depending on whether the DVS event leads or lags the current car position. The amount of advancement or retarding is set by a 'mixing factor' (typically about 0.02) that mixes the DVS event position with the current car position with the mixing factor proportion. The update is done by projecting the vector v_e from current car position to the event onto the track vector v_t connecting the nearest track vertex to the next one along the track. A 2D lookup table (Fig. J.3) maps pixel coordinates to the nearest track vertex, to speed up the search for the nearest track vertex. The car's track velocity is updated when the nearest vertex changes. The car tracker lifetime is managed by a 'mass' that decays away exponentially with time between DVS events and is incremented with each event. If the mass falls below a certain value, the car is considered to be lost and tracking must be reinitialized.

J.2.2 Slot Car Throttle and Braking Hardware

We designed a slot car controller PCB (Fig. J.5) to control the power to up to 4 slot car tracks from a computer over a full-speed USB2.0 interface. This controller controls power to the cars by 1.5 kHz PWM modulation of the 17 V track power, and also can short the car motor across a 20 Ω resistor for electronic braking. Only one track controller is currently used and the other track is controlled by the human using the standard throttle. The controller is updated with polling interval of 1 ms. The full

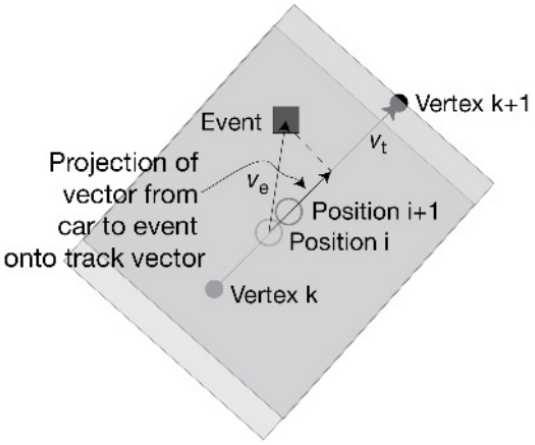


Figure J.4: CarTracker car position update.

PCB design of this controller and its firmware are available in the jAER project.[3] A prototype design did not use optocouplers and the results were frequent resets of the USB microcontroller due to large voltage transients caused by sparking of the car contact to the track which propagated back through the electronics. The final design of Fig. J.5 uses optocouplers to correct this problem.

J.2.3 Throttle Control

Controlling the computer car consists of setting the throttle value or applying the brake at each vertex of the track model based on a vector of throttle/brake settings called a throttle profile. An example throttle profile is shown in Fig. J.6. We investigated a number of methods to optimize the throttle profile. Eventually we found that the fastest method is to 1) determine an initial throttle profile by settings derived automatically from the track curvature so that straight sections have higher initial throttle; 2) examining the car visually and then gradually increasing the throttle or applying brake by eye, using a GUI interface to "paint" throttle and brake settings. An evolutionary method was also developed to learn the optimum throttle profile by inserting throttle increases and seeing if they result in successful laps. After a crash, the insertion is removed or braking points are inserted. The required learning time is currently still considerably longer than by manual adjustment of the profile and more work needs to be done to understand the optimum strategy.

To make racing more competitive, a mode can be enabled that slows the computer down from its optimum throttle value to a minimum value, depending on how far ahead is the computer car. Typically a value of one third of the total track for complete slow-down is effective for resulting in exciting side-by-side racing.

[3] http://sourceforge.net/p/jaer/code/HEAD/tree/devices/pcbs/SlotCarControllerPCB/ (r5522)

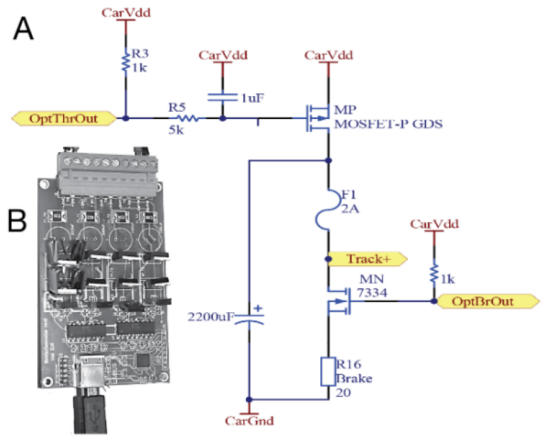


Figure J.5: The slot car controller PCB controls up to 4 lanes.[3] A: interface circuit of a single lane for throttle and brake control. The optocoupler pulldown outputs (OptThrOut & OutBrOut) feed the power MOSFET gates through RC low pass filters with $\tau = 0.5\,\text{ms}$. A fuse F1 protects against shorts. R16 is a power resistor for motor braking. B: Fabricated PCB with USB cable (bottom) and track/power connections (top).

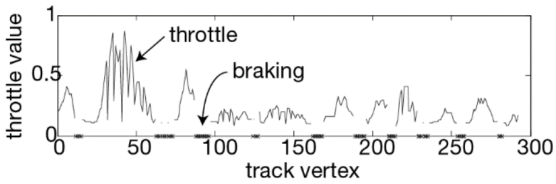


Figure J.6: Throttle and brake profile that achieves 4.1 s lap times on track in Fig. J.2.

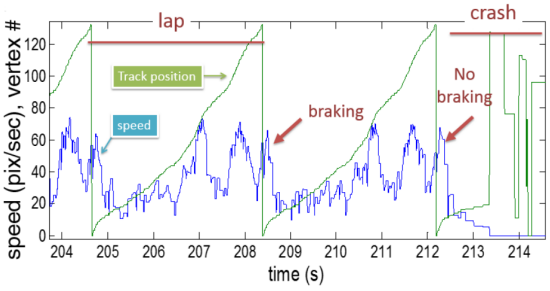


Figure J.7: Slot car position and speed vs. time on a different track. Two laps are completed successfully using motor braking to slow down just after track vertex 0. On the last lap, motor braking is disabled, resulting in a crash.

J.3 Results

A series of YouTube videos document the evolution of the slot car racer since 2010.[4][5][6] The final video shows the setup and a race between computer and human, including control that slows the computer car when it is ahead of the human.

A sample of two recorded laps by the computer controlled car is shown in Fig. J.7. This data is taken from a different track. Over two laps, the car position increases and then wraps back to vertex 0. At the end of the straightway, motor braking rapidly decreases the car speed, as indicated by the "braking" arrow. During the last lap, motor braking is disabled ("No braking"), resulting in a crash after the straightway.

A series of ten 3-lap races between human and computer had the following results: The first human had 1 win, 2 losses, and 7 DNF (did not finish, i.e. crashed). The second human had 3 wins, 4 losses, and 3 DNFs, however after the feedback control to slow the computer car was turned off halfway through the series of ten races, the second human was no longer able to win.

Processing cost and throughput Processing *SlotCarRacer* on a Lenovo W510 Core i7 laptop results in a CPU load of 1 % to 3 % for the Java virtual machine, when graphical rendering is disabled.

The update interval on the host computer was determined by instrumenting the USB data packets received by the high-priority USB processing thread using the Java method *System.nanoTime()*. All the slot car racer processing is done in this thread, rather than the display rendering thread, which runs at most 60 Hz. The DAVIS camera includes a feature called 'early packet timer' which ensures that USB FIFOs are committed to

[4] slotCarRacingTelluride2010.wmv (2010): http://youtu.be/ALneVn-Ls2Q
[5] Slot car racer controlled by DVS Capo Caccia 2014: http://youtu.be/CnGPGiZuFRI
[6] Slot Car Racing with DAVIS neuromorphic vision sensor: http://youtu.be/AsO1TWS8_VA

the host with intervals of at most 1.5 ms, and on the host side the processing intervals closely matched this interval.

J.4 Conclusion

Conventional machine vision using frame-based sensors faces a fundamental latency-power tradeoff. Low latency can only be achieved by processing at a high frame rate, which burns more power. The CPU load of less than 3 % achieved in the slot car racer is a result of the low data rate averaging 5keps (thousand events per second) per car. The staring camera scenario is ideal for using the DVS, since only the small moving slot cars create DVS events. The early packet timer transmits available events from the camera at a minimum rate of $1/1.5\,\text{ms} = 666\,\text{Hz}$, which means that most packets sent to the host contain only about 7 events per car. This is a small amount of data to process. By comparison, if the 240x180 pixel image could be transmitted to the host at 666 Hz, it would mean a data rate of 29M pixels/s, which would be a factor of about 1,000 times more data.

The slot car racer robot is a popular demonstration of the use of a DAVIS sensor, mainly because racing is fun and it is a contest between human and computer that involves quick reaction times. The principle of operation is simple and easy to explain. In practice, because the computer is so precise, and because it can use motor braking, it is practically unbeatable and so to produce the illusion of a competitive race it is necessary to enable the mode where the computer car is slowed down if it is ahead.

The control of the car is in some sense open-loop because the throttle and brake are applied according to the instantaneous position of the car on the track, regardless of the car's speed. Future enhancements could focus on developing an adaptive model-based controller that regulates the speed of the car to a desired level that is safe for the curvature. Our attempts to do this were not successful because the model is surprisingly complex. The physics of car movement along the track and the action of the power applied to the car on its speed are complicated by track curvature, friction, individual car variability, motor heating, etc. However the short latency ofsensor measurement could enable visual feedback on throttle control.

Bibliography

[1] E. H. Adelson and J. R. Bergen. "Spatiotemporal energy models for the perception of motion". In: *J. Opt. Soc. Am. A* 2.2 (1985), pp. 284–299. DOI: 10.1364/JOSAA.2.000284.

[2] A. Agarwal, K. Mierle, et al. *Ceres Solver*. http://ceres-solver.org.

[3] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha. "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip". In: *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 34.10 (2015), pp. 1537–1557. ISSN: 0278-0070. DOI: 10.1109/TCAD.2015.2474396.

[4] H. S. Alismail, L. D. Baker, and B. Browning. "Continuous trajectory estimation for 3D SLAM from actuated lidar". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2014, pp. 6096–6101. DOI: 10.1109/ICRA.2014.6907757.

[5] S. Anderson, F. Dellaert, and T. D. Barfoot. "A hierarchical wavelet decomposition for continuous-time SLAM". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2014, pp. 373–380. DOI: 10.1109/ICRA.2014.6906884.

[6] P. Bardow, A. J. Davison, and S. Leutenegger. "Simultaneous Optical Flow and Intensity Estimation From an Event Camera". In: *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.* 2016. DOI: 10.1109/CVPR.2016.102.

[7] F. Barranco, C. Fermuller, Y. Aloimonos, and T. Delbruck. "A Dataset for Visual Navigation with Neuromorphic Methods". In: *Front. Neurosci.* 10 (2016), p. 49. DOI: 10.3389/fnins.2016.00049.

[8] A. N. Belbachir. *Smart Cameras*. Springer US, 2009. ISBN: 9781441909534.

[9] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi. "Event-Based Visual Flow". In: *IEEE Trans. Neural Netw. Learn. Syst.* 25.2 (2014), pp. 407–417. DOI: 10.1109/TNNLS.2013.2273537.

[10] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. Srinivasan. "Asynchronous frameless event-based optical flow". In: *Neural Netw.* 27 (2012), pp. 32–37. DOI: 10.1016/j.neunet.2011.11.001.

[11] R. Benosman, S.-H. Ieng, P. Rogister, and C. Posch. "Asynchronous Event-Based Hebbian Epipolar Geometry". In: *IEEE Trans. Neural Netw.* 22.11 (2011), pp. 1723–1734. DOI: 10.1109/TNN.2011.2167239.

[12] P. J. Besl and N. D. McKay. "A method for registration of 3-D shapes". In: *IEEE Trans. Pattern Anal. Machine Intell.* 14.2 (1992), pp. 239–256. DOI: 10.1109/34.121791.

[13] C. Bibby and I. D. Reid. "A hybrid SLAM representation for dynamic marine environments". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2010, pp. 257–264. DOI: 10.1109/ROBOT.2010.5509262.

[14] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.

[15] K. Boahen. "A Retinomorphic Chip with Parallel Pathways: Encoding INCREASING, ON, DECREASING, and OFF Visual Signals". In: *Analog Integr. Circuits Signal Process.* 30.2 (2002), pp. 121–135. DOI: 10.1023/A:1013751627357.

[16] A. Bolopion, Z. Ni, J. Agnus, R. Benosman, and S. Régnier. "Stable Haptic Feedback based on a Dynamic Vision Sensor for Microrobotics". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2012. DOI: 10.1109/IROS.2012.6385557.

[17] J.-Y. Bouguet. *Camera Calibration Toolbox for Matlab*. URL: http://www.vision.caltech.edu/bouguetj/calib_doc/.

[18] C. Braendli, J. Strubel, S. Keller, D. Scaramuzza, and T. Delbruck. "ELiSeD - An Event-Based Line Segment Detector". In: *Int. Conf. Event-Based Control, Comm. Signal Proc. (EBCCSP)*. Krakow, Poland, June 2016. DOI: 10.1109/EBCCSP.2016.7605244.

[19] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck. "A 240x180 130dB 3us Latency Global Shutter Spatiotemporal Vision Sensor". In: *IEEE J. Solid-State Circuits* 49.10 (2014), pp. 2333–2341. ISSN: 0018-9200. DOI: 10.1109/JSSC.2014.2342715.

[20] C. Brandli, L. Muller, and T. Delbruck. "Real-time, high-speed video decompression using a frame- and event-based DAVIS sensor". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. June 2014, pp. 686–689. DOI: 10.1109/ISCAS.2014.6865228.

[21] W. G. Breckenridge. *Quaternions proposed standard conventions*. Tech. rep. NASA Jet Propulsion Laboratory, Oct. 1979.

[22] D. Brescianini, M. Hehn, and R. D'Andrea. "Quadrocopter Pole Acrobatics". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. Nov. 2013, pp. 3472–3479. DOI: 10.1109/IROS.2013.6696851.

[23] J. Canny. "A Computational Approach to Edge Detection". In: *IEEE Trans. Pattern Anal. Machine Intell.* PAMI-8.6 (Nov. 1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[24] J. Carneiro, S.-H. Ieng, C. Posch, and R. Benosman. "Event-based 3D reconstruction from neuromorphic retinas". In: *Neural Netw.* 45 (2013), pp. 27–38. DOI: 10.1016/j.neunet.2013.03.006.

[25] S. Caviglia, L. Pinna, M. Valle, and C. Bartolozzi. "Spike-Based Readout of POSFET Tactile Sensors". In: *IEEE Trans. Circuits Syst. I, Reg. Papers* PP.99 (2016), pp. 1–11. DOI: 10.1109/TCSI.2016.2561818.

[26] A. Censi and D. Scaramuzza. "Low-Latency Event-Based Visual Odometry". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2014. DOI: 10.1109/IROS.2016.7758089.

[27] 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 Int. Conf. Intell. Robot. Syst. (IROS)*. 2013. DOI: 10.1109/IROS.2013.6696456.

[28] X. Clady, C. Clercq, S.-H. Ieng, F. Houseini, M. Randazzo, L. Natale, C. Bartolozzi, and R. Benosman. "Asynchronous visual event-based time-to-contact". In: *Front. Neurosci.* 8.9 (2014). DOI: 10.3389/fnins.2014.00009.

[29] X. Clady, S.-H. Ieng, and R. Benosman. "Asynchronous event-based corner detection and matching". In: *Neural Netw.* 66 (2015), pp. 91–106. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2015.02.013.

[30] C. Tomasi and T. Kanade. *Detection and Tracking of Point Features*. Tech. rep. Carnegie Mellon University, 1991.

[31] 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 Int. Symp. Circuits Syst. (ISCAS)*. 2009, pp. 781–784. DOI: 10.1109/ISCAS.2009.5117867.

[32] M. Cook, L. Gugelmann, F. Jug, C. Krautz, and A. Steger. "Interacting maps for fast visual interpretation". In: *Int. Joint Conf. Neural Netw. (IJCNN)*. 2011, pp. 770–776. DOI: 10.1109/IJCNN.2011.6033299.

[33] T. Delbruck. "Frame-free dynamic digital vision". In: *Proc. Int. Symp. Secure-Life Electron.* 2008, pp. 21–26.

[34] T. Delbruck and M. Lang. "Robotic Goalie with 3ms Reaction Time at 4% CPU Load Using Event-Based Dynamic Vision Sensor". In: *Front. Neurosci.* 7.223 (2013). DOI: 10.3389/fnins.2013.00223.

[35] T. Delbruck and P. Lichtsteiner. "Fast sensory motor control based on event-based hybrid neuromorphic-procedural system". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. May 2007, pp. 845–848. DOI: 10.1109/ISCAS.2007.378038.

[36] T. Delbruck, B. Linares-Barranco, E. Culurciello, and C. Posch. "Activity-driven, event-based vision sensors". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. May 2010, pp. 2426–2429. DOI: 10.1109/ISCAS.2010.5537149.

[37] T. Delbruck, M. Pfeiffer, R. Juston, G. Orchard, E. Müggler, A. Linares-Barranco, and M. W. Tilden. "Human vs. Computer Slot Car Racing using an Event and Frame-Based DAVIS Vision Sensor". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. Lisbon, Portugal, May 2015, pp. 2409–2412. DOI: 10.1109/ISCAS.2015.7169170.

[38] T. Delbruck, V. Villanueva, and L. Longinotti. "Integration of dynamic vision sensor with inertial measurement unit for electronically stabilized event-based vision". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. June 2014, pp. 2636–2639. DOI: 10.1109/ISCAS.2014.6865714.

[39] J. Delmerico, A. Giusti, E. Mueggler, L. M. Gambardella, and D. Scaramuzza. ""On-the-spot Training" for Terrain Classification in Autonomous Air-Ground Collaborative Teams". In: *Int. Symp. Experimental Robotics (ISER)*. 2016. DOI: 10.1007/978-3-319-50115-4_50.

[40] J. Delmerico, E. Mueggler, J. Nitsch, and D. Scaramuzza. "Active Autonomous Aerial Exploration for Ground Robot Path Planning". In: *IEEE Robot. Autom. Lett.* 2.2 (2017), pp. 664–671. DOI: 10.1109/LRA.2017.2651163.

[41] D. Drazen, P. Lichtsteiner, P. Häfliger, T. Delbrück, and A. Jensen. "Toward real-time particle tracking using an event-based dynamic vision sensor". In: *Experiments in Fluids* 51.5 (2011), pp. 1465–1469. ISSN: 0723-4864. DOI: 10.1007/s00348-011-1207-y.

[42] D. Drubach. *The Brain Explained*. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.

[43] R. O. Duda and P. E. Hart. "Use of the Hough Transformation to Detect Lines and Curves in Pictures". In: *Commun. ACM* 15.1 (1972), pp. 11–15. DOI: 10.1145/361237.361242.

[44] P. Dudek and P. J. Hicks. "A general-purpose processor-per-pixel analog SIMD vision chip". In: *IEEE Trans. Circuits Syst. I, Reg. Papers* 52.1 (2005), pp. 13–20. DOI: 10.1109/TCSI.2004.840093.

[45] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza. "Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor MAV". In: *J. Field Robot.* 33.4 (2016), pp. 431–450. ISSN: 1556-4967. DOI: 10.1002/rob.21581.

[46] M. Faessler, E. Mueggler, K. Schwabe, and D. Scaramuzza. "A Monocular Pose Estimation System based on Infrared LEDs". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2014, pp. 907–913. DOI: 10.1109/ICRA.2014.6906962.

[47] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza. "Aggressive Quadrotor Flight through Narrow Gaps with Onboard Sensing and Computing". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017.

[48] M. A. Fischler and R. C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Commun. ACM* 24.6 (1981), pp. 381–395. ISSN: 0001-0782. DOI: http://doi.acm.org/10.1145/358669.358692.

[49] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. "On-Manifold Preintegration for Real-Time Visual-Inertial Odometry". In: *IEEE Trans. Robot.* 33.1 (2017), pp. 1–21. DOI: 10.1109/TRO.2016.2597321.

[50] C. Forster, M. Pizzoli, and D. Scaramuzza. "SVO: Fast Semi-Direct Monocular Visual Odometry". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2014, pp. 15–22. DOI: 10.1109/ICRA.2014.6906584.

[51] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown. "Overview of the SpiNNaker System Architecture". In: *IEEE Trans. Comput.* 62.12 (2013), pp. 2454–2467. ISSN: 0018-9340. DOI: 10.1109/TC.2012.142.

[52] P. Furgale, T. D. Barfoot, and G. Sibley. "Continuous-Time Batch Estimation using Temporal Basis Functions". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2012, pp. 2088–2095. DOI: 10.1109/ICRA.2012.6225005.

[53] G. Gallego, J. E. A. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza. "Event-based, 6-DOF Camera Tracking for High-Speed Applications". arXiv:1607.03468. 2016.

[54] G. Gallego, J. E. A. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza. "Event-based, 6-DOF Camera Tracking for High-Speed Applications". In: *IEEE Trans. Pattern Anal. Machine Intell.* (2017). under review.

[55] G. Gallego and D. Scaramuzza. "Accurate Angular Velocity Estimation with an Event Camera". In: *IEEE Robot. Autom. Lett.* 2 (2 2017), pp. 632–639. ISSN: 2377-3766. DOI: 10.1109/LRA.2016.2647639.

[56] T. A. Gibson, S. Heath, R. P. Quinn, A. H. Lee, J. T. Arnold, T. S. Sonti, A. Whalley, G. P. Shannon, B. T. Song, J. A. Henderson, and J. Wiles. "Event-Based Visual Data Sets for Prediction Tasks in Spiking Neural Networks". In: *Int. Conf. Artificial Neural Netw.* 2014. DOI: 10.1007/978-3-319-11179-7_80.

[57] S. Grzonka, G. Grisetti, and W. Burgard. "A Fully Autonomous Indoor Quadrotor". In: *IEEE Trans. Robot.* 28.1 (2012), pp. 90–100. DOI: 10.1109/TRO.2011.2162999.

[58] C. Harris and M. Stephens. "A combined corner and edge detector". In: *Proc. Fourth Alvey Vision Conf.* Vol. 15. Manchester, UK, 1988, pp. 147–151.

[59] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Second Edition. Cambridge University Press, 2003.

[60] R. Hoffmann, D. Weikersdorfer, and J. Conradt. "Autonomous indoor exploration with an event-based visual SLAM system". In: *Eur. Conf. Mobile Robots (ECMR)*. 2013, pp. 38–43. DOI: 10.1109/ECMR.2013.6698817.

[61] Y. Hu, H. Liu, M. Pfeiffer, and T. Delbruck. "DVS Benchmark Datasets for Object Tracking, Action Recognition, and Object Recognition". In: *Front. Neurosci.* 10 (2016), p. 405. DOI: 10.3389/fnins.2016.00405.

[62] D. Q. Huynh. "Metrics for 3D Rotations: Comparison and Analysis". In: *J. Math. Imaging Vis.* 35.2 (2009), pp. 155–164. DOI: 10.1007/s10851-009-0161-2.

[63] R. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *J. Basic Eng.* 82 (1 1960), pp. 35–45.

[64] R. Käslin, P. Fankhauser, E. Stumm, Z. Taylor, E. Mueggler, J. Delmerico, D. Scaramuzza, R. Siegwart, and M. Hutter. "Collaborative localization of aerial and ground robots through elevation maps". In: *IEEE Int. Symp. Safety, Security, and Rescue Robot. (SSRR)*. Oct. 2016, pp. 284–290. DOI: 10.1109/SSRR.2016.7784317.

[65] C. Kerl, J. Stückler, and D. Cremers. "Dense Continuous-Time Tracking and Mapping with Rolling Shutter RGB-D Cameras". In: *Int. Conf. Comput. Vis. (ICCV)*. 2015. DOI: 10.1109/ICCV.2015.261.

[66] H. Kim, A. Handa, R. Benosman, S.-H. Ieng, and A. J. Davison. "Simultaneous Mosaicing and Tracking with an Event Camera". In: *British Machine Vis. Conf. (BMVC)*. 2014. DOI: 10.5244/C.28.26.

[67] H. Kim, S. Leutenegger, and A. J. Davison. "Real-Time 3D Reconstruction and 6-DoF Tracking with an Event Camera". In: *Eur. Conf. Comput. Vis. (ECCV)*. 2016, pp. 349–364. DOI: 10.1007/978-3-319-46466-4_21.

[68] L. Kneip, D. Scaramuzza, and R. Siegwart. "A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation". In: *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.* 2011, pp. 2969–2976. DOI: 10.1109/CVPR.2011.5995464.

[69] J. Kogler, C. Sulzbachner, and W. Kubinger. "Bio-inspired Stereo Vision System with Silicon Retina Imagers". In: *Int. Conf. Comput. Vis. Syst. (ICVS)*. 2009, pp. 174–183. DOI: 10.1007/978-3-642-04667-4_18.

[70] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza. "Low-latency Visual Odometry using Event-based Feature Tracks". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. Daejeon, Korea, Oct. 2016, pp. 16–23. DOI: 10.1109/IROS.2016.7758089.

[71] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *Ann. Math. Statist.* 22.1 (1951), pp. 79–86. DOI: 10.1214/aoms/1177729694.

[72] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat, and R. Benosman. "Asynchronous Event-Based Multikernel Algorithm for High-Speed Visual Features Tracking". In: *IEEE Trans. Neural Netw. Learn. Syst.* 26.8 (Aug. 2015), pp. 1710–1720. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2014.2352401.

[73] D. N. Lee. "A theory of visual control of braking based on information about time-to-collision." In: *Perception* 5.4 (1976), pp. 437–459. DOI: 10.1068/p050437.

[74] J. Lee, T. Delbruck, P. K. J. Park, M. Pfeiffer, C.-W. Shin, H. Ryu, and B. C. Kang. "Live demonstration: Gesture-Based remote control using stereo pair of dynamic vision sensors". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. 2012. DOI: 10.1109/ISCAS.2012.6272144.

[75] C. Li, C. Brandli, R. Berner, H. Liu, M. Yang, S.-C. Liu, and T. Delbruck. "An RGBW Color VGA Rolling and Global Shutter Dynamic and Active-Pixel Vision Sensor". In: *International Image Sensor Workshop (IISW)*. Vaals, Netherlands, June 2015.

[76] C. Li, C. Brandli, R. Berner, H. Liu, M. Yang, S.-C. Liu, and T. Delbruck. "Design of an RGBW color VGA rolling and global shutter dynamic and active-pixel vision sensor". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. 2015. DOI: 10.1109/ISCAS.2015.7168734.

[77] P. Lichtsteiner, C. Posch, and T. Delbruck. "A 128×128 120 dB 15 μs latency asynchronous temporal contrast vision sensor". In: *IEEE J. Solid-State Circuits* 43.2 (2008), pp. 566–576. DOI: 10.1109/JSSC.2007.914337.

[78] P. Lichtsteiner, C. Posch, and T. Delbruck. "A 128x128 120dB 30mW asynchronous vision sensor that responds to relative intensity change". In: *IEEE Intl. Solid-State Circuits Conf. (ISSCC)*. Feb. 2006, pp. 2060–2069. DOI: 10.1109/ISSCC.2006.1696265.

[79] M. Litzenberger, A. N. Belbachir, N. Donath, G. Gritsch, H. Garn, B. Kohn, C. Posch, and S. Schraml. "Estimation of Vehicle Speed Based on Asynchronous Data from a Silicon Retina Optical Sensor". In: *IEEE Intell. Transp. Sys. Conf.* Sept. 2006, pp. 653–658. DOI: 10.1109/ITSC.2006.1706816.

[80] M. Litzenberger, C. Posch, D. Bauer, A. N. Belbachir, P. Schön, B. Kohn, and H. Garn. "Embedded Vision System for Real-Time Object Tracking using an Asynchronous Transient Vision Sensor". In: *Digital Signal Processing Workshop*. Sept. 2006, pp. 173–178. DOI: 10.1109/DSPWS.2006.265448.

[81] H. Liu, D. P. Moeys, G. Das, D. Neil, S.-C. Liu, and T. Delbruck. "Combined frame- and event-based detection and tracking". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. 2016, pp. 2511–2514. DOI: 10.1109/ISCAS.2016.7539103.

[82] S.-C. Liu and T. Delbruck. "Neuromorphic sensory systems". In: *Current Opinion in Neurobiology* 20.3 (2010), pp. 288–295. DOI: 10.1016/j.conb.2010.03.007.

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

[84] S.-C. Liu, A. van Schaik, B. A. Minch, and T. Delbruck. "Asynchronous Binaural Spatial Audition Sensor With 2x64x4 Channel Output". In: *IEEE Trans. Biomed. Circuits Syst.* 8.4 (2014), pp. 453–464. DOI: 10.1109/TBCAS.2013.2281834.

[85] B. D. Lucas and T. Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *Int. Joint Conf. Artificial Intell.* 1981, pp. 674–679.

[86] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea. "A platform for aerial robotics research and demonstration: The Flying Machine Arena". In: *J. Mechatronics* 24.1 (Feb. 2014), pp. 41–54. DOI: 10.1016/j.mechatronics.2013.11.006.

[87] S. Lupashin, A. Schöllig, M. Sherback, and R. D'Andrea. "A simple learning strategy for high-speed quadrocopter multi-flips". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. May 2010, pp. 1642–1648. DOI: 10.1109/ROBOT.2010.5509452.

[88] Y. Ma, S. Soatto, J. Košecká, and S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer, 2004.

[89] M. Mahowald. "The Silicon Retina". In: *An Analog VLSI System for Stereoscopic Vision*. Boston, MA: Springer US, 1994, pp. 4–65. ISBN: 978-1-4615-2724-4. DOI: 10.1007/978-1-4615-2724-4_2.

[90] M. Mahowald. "VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function". PhD thesis. Pasadena, California: California Institute of Technology, May 1992.

[91] C. A. Mead and M. Mahowald. "A silicon model of early visual processing". In: *Neural Netw.* 1.1 (1989), pp. 91–97. DOI: 10.1016/0893-6080(88)90024-X.

[92] D. Mellinger and V. Kumar. "Minimum snap trajectory generation and control for quadrotors". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. May 2011, pp. 2520–2525. DOI: 10.1109/ICRA.2011.5980409.

[93] D. Mellinger, N. Michael, and V. Kumar. "Trajectory generation and control for precise aggressive maneuvers with quadrotors". In: *Int. J. Robot. Research* 31.5 (2012), pp. 664–674. DOI: 10.1177/0278364911434236.

[94] G. Metta, L. Natale, F. Nori, G. Sandini, D. Vernon, L. Fadiga, C. von Hofsten, K. Rosander, M. Lopes, J. Santos-Victor, A. Bernardino, and L. Montesano. "The iCub humanoid robot: An open-systems platform for research in cognitive development". In: *Neural Netw.* 23 (8–9 2010), pp. 1125–1134. DOI: 10.1016/j.neunet.2010.08.010.

[95] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbruck. "Steering a Predator Robot using a Mixed Frame/Event-Driven Convolutional Neural Network". In: *Int. Conf. Event-Based Control, Comm. Signal Proc. (EBCCSP)*. 2016. DOI: 10.1109/EBCCSP.2016.7605233.

[96] E. Mueggler, C. Bartolozzi, and D. Scaramuzza. "Fast Event-based Corner Detection". In: *British Machine Vis. Conf. (BMVC)*. 2017.

[97] E. Mueggler, N. Baumli, F. Fontana, and D. Scaramuzza. "Towards Evasive Maneuvers with Quadrotors using Dynamic Vision Sensors". In: *Eur. Conf. Mobile Robots (ECMR)*. 2015, pp. 1–8. DOI: 10.1109/ECMR.2015.7324048.

[98] E. Mueggler, M. Faessler, F. Fontana, and D. Scaramuzza. "Aerial-guided Navigation of a Ground Robot among Movable Obstacles". In: *IEEE Int. Symp. Safety, Security, and Rescue Robot. (SSRR)*. 2014, pp. 1–8. DOI: 10.1109/SSRR.2014.7017662.

[99] E. Mueggler, C. Forster, N. Baumli, G. Gallego, and D. Scaramuzza. "Lifetime Estimation of Events from Dynamic Vision Sensors". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2015, pp. 4874–4881. DOI: 10.1109/ICRA.2015.7139876.

[100] E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza. "Continuous-Time Visual-Inertial Trajectory Estimation with Event Cameras". In: *IEEE Trans. Robot.* (2017). under review.

[101] E. Mueggler, G. Gallego, and D. Scaramuzza. "Continuous-Time Trajectory Estimation for Event-based Vision Sensors". In: *Robotics: Science and Systems (RSS)*. 2015. DOI: 10.15607/RSS.2015.XI.036.

[102] E. Mueggler, B. Huber, and D. Scaramuzza. "Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2014, pp. 2761–2768. DOI: 10.1109/IROS.2014.6942940.

[103] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza. "The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM". In: *Int. J. Robot. Research* 36 (2 2017), pp. 142–149. DOI: 10.1177/0278364917691115.

[104] E. Mueller, A. Censi, and E. Frazzoli. "Low-latency Heading Feedback Control with Neuromorphic Vision Sensors using Efficient Approximated Incremental Inference". In: *IEEE Conf. Decision Control (CDC)*. 2015. DOI: 10.1109/CDC.2015.7402002.

[105] M. Mueller, S. Lupashin, and R. D'Andrea. "Quadrocopter ball juggling". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2011, pp. 4972–4978. DOI: 10.1109/IROS.2012.6385963.

[106] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. "DTAM: Dense Tracking and Mapping in Real-Time". In: *Int. Conf. Comput. Vis. (ICCV)*. Nov. 2011, pp. 2320–2327.

[107] Z. Ni, C. Pacoret, R. Benosman, S. Ieng, and S. Regnier. "Asynchronous event-based high speed vision for microparticle tracking". In: *Journal of Microscopy* 245.3 (2012), pp. 236–244. DOI: 10.1111/j.1365-2818.2011.03565.x.

[108] Z. Ni, A. Bolopion, J. Agnus, R. Benosman, and S. Regnier. "Asynchronous Event-Based Visual Shape Tracking for Stable Haptic Feedback in Microrobotics". In: *IEEE Trans. Robot.* 28 (5 2012), pp. 1081–1089. DOI: 10.1109/TRO.2012.2198930.

[109] Z. Ni, S.-H. Ieng, C. Posch, S. Régnier, and R. Benosman. "Visual Tracking Using Neuromorphic Asynchronous Event-Based Cameras". In: *Neural Computation* 27 (4 2015), pp. 925–953. DOI: 10.1162/NECO_a_00720.

[110] D. Nistér. "An efficient solution to the five-point relative pose problem". In: *IEEE Trans. Pattern Anal. Machine Intell.* 26.6 (2004), pp. 756–777. DOI: 10.1109/TPAMI.2004.17.

[111] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor. "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades". In: *Front. Neurosci.* 9 (2015), p. 437. DOI: 10.3389/fnins.2015.00437.

[112] A. Patron-Perez, S. Lovegrove, and G. Sibley. "A Spline-Based Trajectory Representation for Sensor Fusion and Rolling Shutter Cameras". In: *Int. J. Comput. Vis.* 113.3 (2015), pp. 208–219. DOI: 10.1007/s11263-015-0811-3.

[113] E. Piatkowska, A. N. Belbachir, S. Schraml, and M. Gelautz. "Spatiotemporal multiple persons tracking using Dynamic Vision Sensor". In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. Workshop*. June 2012, pp. 35–40. DOI: 10.1109/CVPRW.2012.6238892.

[114] M. Pizzoli, C. Forster, and D. Scaramuzza. "REMODE: Probabilistic, Monocular Dense Reconstruction in Real Time". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2014, pp. 2609–2616. URL: http://dx.doi.org/10.1109/ICRA.2014.6907233.

[115] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat. "Comparing ICP Variants on Real-World Data Sets". In: *Auton. Robots* 34.3 (Feb. 2013), pp. 133–148. DOI: 10.1007/s10514-013-9327-2.

[116] 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 J. Solid-State Circuits* 46.1 (Jan. 2011), pp. 259–275. ISSN: 0018-9200. DOI: 10.1109/JSSC.2010.2085952.

[117] C. Posch, D. Matolin, and R. Wohlgenannt. "A QVGA 143dB dynamic range asynchronous address-event PWM dynamic image sensor with lossless pixel-level video compression". In: *IEEE Intl. Solid-State Circuits Conf. (ISSCC)*. Feb. 2010, pp. 400–401. DOI: 10.1109/ISSCC.2010.5433973.

[118] C. Posch, D. Matolin, and R. Wohlgenannt. "An asynchronous time-based image sensor". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. May 2008, pp. 2130–2133. DOI: 10.1109/ISCAS.2008.4541871.

[119] X. Qi, X. Guo, and J. Harris. "A time-to-first spike CMOS imager". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. 2004. DOI: 10.1109/ISCAS.2004.1329131.

[120] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri. "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses". In: *Front. Neurosci.* 9 (2015), p. 141. ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00141.

[121] K. Qin. "General matrix representations for B-splines". In: *The Visual Computer* 16.3–4 (2000), pp. 177–186.

[122] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. "ROS: an open-source Robot Operating System". In: *ICRA Workshop Open Source Softw.* Vol. 3. 2. 2009, p. 5.

[123] H. Rebecq, G. Gallego, and D. Scaramuzza. "EMVS: Event-based Multi-View Stereo". In: *British Machine Vis. Conf. (BMVC)*. Sept. 2016.

[124] H. Rebecq, T. Horstschäfer, G. Gallego, and D. Scaramuzza. "EVO: A Geometric Approach to Event-based 6-DOF Parallel Tracking and Mapping in Real-Time". In: *IEEE Robot. Autom. Lett.* 2 (2 2017), pp. 593–600. ISSN: 2377-3766. DOI: 10.1109/LRA.2016.2645143.

[125] C. Reinbacher, G. Graber, and T. Pock. "Real-Time Intensity-Image Reconstruction for Event Cameras Using Manifold Regularisation". In: *British Machine Vis. Conf. (BMVC)*. 2016.

[126] E. Rosten and T. Drummond. "Machine learning for high-speed corner detection". In: *Eur. Conf. Comput. Vis. (ECCV)*. Graz, Austria, 2006, pp. 430–443. DOI: 10.1007/11744023_34.

[127] B. Rueckauer and T. Delbruck. "Evaluation of Event-Based Algorithms for Optical Flow with Ground-Truth from Inertial Measurement Sensor". In: *Front. Neurosci.* 10.176 (2016). DOI: 10.3389/fnins.2016.00176.

[128] P.-F. Rüedi, P. Heim, F. Kaess, E. Grenet, F. Heitger, P.-Y. Burgi, S. Gyger, and P. Nussbaum. "A 128x128 pixel 120-dB dynamic-range vision-sensor chip for image contrast and orientation extraction". In: *IEEE J. Solid-State Circuits* 38.12 (2003), pp. 2325–2333. DOI: 10.1109/JSSC.2003.819169.

[129] S. Schraml, A. N. Belbachir, and H. Bischof. "Event-driven stereo matching for real-time 3D panoramic vision". In: *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.* June 2015, pp. 466–474. DOI: 10.1109/CVPR.2015.7298644.

[130] S. Schraml, A. N. Belbachir, N. Milosevic, and P. Schön. "Dynamic Stereo Vision System for Real-time Tracking". In: *IEEE Int. Symp. Circuits Syst. (ISCAS)*. 2010, pp. 1409–1412. DOI: 10.1109/ISCAS.2010.5537289.

[131] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gomez-Rodriguez, L. Camunas-Mesa, R. Berner, M. Rivas-Perez, T. Delbruck, S.-C. Liu, R. Douglas, P. Hafliger, G. Jimenez-Moreno, A. C. Ballcels, T. Serrano-Gotarredona, A. J. Acosta-Jimenez, and B. Linares-Barranco. "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory-Processing-Learning-Actuating System for High-Speed Visual Object Recognition and Tracking". In: *IEEE Trans. Neural Netw.* 20.9 (2009), pp. 1417–1438. DOI: 10.1109/TNN.2009.2023653.

[132] S. Shen, N. Michael, and V. Kumar. "Autonomous multi-floor indoor navigation with a computationally constrained MAV". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. May 2011, pp. 20–25. DOI: 10.1109/ICRA.2011.5980357.

[133] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar. "Vision-based state estimation and trajectory control towards aggressive flight with a quadrotor". In: *Robotics: Science and Systems (RSS)*. June 2013. DOI: 10.15607/RSS.2013.IX.03.

[134] J. Shi and C. Tomasi. "Good features to track". In: *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recog.* June 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[135] B. Son, Y. Suh, S. Kim, H. Jung, J.-S. Kim, C. Shin, K. Park, K. Lee, J. Park, J. Woo, Y. Roh, H. Lee, Y. Wang, I. Ovsiannikov, and H. Ryu. "A 640x480 dynamic vision sensor with a 9um pixel and 300Meps address-event representation". In: *IEEE Intl. Solid-State Circuits Conf. (ISSCC)*. 2017. DOI: 10.1109/ISSCC.2017.7870263.

[136] R. Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer, 2010. ISBN: 9781848829343.

[137] C. Tan, S. Lallee, and G. Orchard. "Benchmarking neuromorphic vision: lessons learnt from computer vision". In: *Front. Neurosci.* 9 (2015), p. 374. ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00374.

[138] D. Tedaldi, G. Gallego, E. Mueggler, and D. Scaramuzza. "Feature Detection and Tracking with the Dynamic and Active-pixel Vision Sensor (DAVIS)". In: *Int. Conf. Event-Based Control, Comm. Signal Proc. (EBCCSP)*. Krakow, Poland, June 2016, pp. 1–7. DOI: 10.1109/EBCCSP.2016.7605086.

[139] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, Cambridge, MA, 2005.

[140] R. Y. Tsai and R. K. Lenz. "A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration". In: *IEEE Trans. Robot.* 5.3 (1989), pp. 345–358. DOI: 10.1109/70.34770.

[141] D. R. Valeiras, G. Orchard, S.-H. Ieng, and R. B. Benosman. "Neuromorphic Event-Based 3D Pose Estimation". In: *Front. Neurosci.* 9 (2016), p. 522. DOI: 10.3389/fnins.2015.00522.

[142] V. Vasco, A. Glover, and C. Bartolozzi. "Fast event-based Harris corner detection exploiting the advantages of event-driven cameras". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2016. DOI: 10.1109/IROS.2016.7759610.

[143] G. Vogiatzis and C. Hernández. "Video-based, Real-Time Multi View Stereo". In: *Image Vis. Comput.* 29.7 (2011), pp. 434–441. DOI: 10.1016/j.imavis.2011.01.006.

[144] D. Weikersdorfer, D. B. Adrian, D. Cremers, and J. Conradt. "Event-based 3D SLAM with a depth-augmented dynamic vision sensor". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. June 2014, pp. 359–364. DOI: 10.1109/ICRA.2014.6906882.

[145] D. Weikersdorfer and J. Conradt. "Event-based Particle Filtering for Robot Self-Localization". In: *IEEE Int. Conf. Robot. Biomimetics (ROBIO)*. 2012, pp. 866–870. DOI: 10.1109/ROBIO.2012.6491077.

[146] D. Weikersdorfer, R. Hoffmann, and J. Conradt. "Simultaneous Localization and Mapping for event-based Vision Systems". In: *Int. Conf. Comput. Vis. Syst. (ICVS)*. 2013, pp. 133–142. DOI: 10.1007/978-3-642-39402-7_14.

[147] S. Weiss, M. W. Achtelik, S. Lynen, M. C. Achtelik, L. Kneip, M. Chli, and R. Siegwart. "Monocular Vision for Long-term Micro Aerial Vehicle State Estimation: A Compendium". In: *J. Field Robot.* 30.5 (Aug. 2013), pp. 803–831. DOI: 10.1002/rob.21466.

[148] M. Yang, S.-C. Liu, and T. Delbruck. "A Dynamic Vision Sensor With 1% Temporal Contrast Sensitivity and In-Pixel Asynchronous Delta Modulator for Event Encoding". In: *IEEE J. Solid-State Circuits* 50.9 (2015), pp. 2149–2160. DOI: 10.1109/JSSC.2015.2425886.

[149] W. Yuan and S. Ramalingam. "Fast Localization and Tracking using Event Sensors". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2016, pp. 4564–4571. DOI: 10.1109/ICRA.2016.7487657.

[150] Z. Zhang. "A Flexible New Technique for Camera Calibration". In: *IEEE Trans. Pattern Anal. Machine Intell.* 22.11 (Nov. 2000), pp. 1330–1334. ISSN: 0162-8828. DOI: 10.1109/34.888718.

[151] A. Z. Zhu, N. Atanasov, and K. Daniilidis. "Event-Based Feature Tracking with Probabilistic Data Association". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017.

Elias Mueggler

Curriculum Vitae

Andreasstrasse 15, AND 2.16
8050 Zurich, Switzerland
☎ *+41 44 635 43 41*
✉ *mueggler@ifi.uzh.ch*

Education

2012–2017 **Ph.D. Student in Computer Science**, *University of Zurich*, Switzerland.
Advisor: Prof. Dr. Davide Scaramuzza
Committee: Prof. Dr. Tobi Delbruck, Prof. Dr. Kostas Daniilidis

2013 **Teaching Certificate in Mechanical Engineering**, *ETH Zurich*, Switzerland.
24 ECTS didactics program including a teaching internship

2012 **Master of Science in Mechanical Engineering**, *ETH Zurich*, Switzerland.
Advisor: Prof. Dr. Raffaello D'Andrea
GPA 5.69/6.0, Year Average: 5.43, Standard Deviation: 0.28

2009 **Bachelor of Science in Mechanical Engineering**, *ETH Zurich*, Switzerland.
GPA 5.22/6.0, Year Average: 4.92, Standard Deviation: 0.32

Exchange

Jan–Mar 2017 **Visiting PhD Student at the iCub Facility**, *Italian Institute of Technology (IIT)*, Genoa, Italy.

Jan–Jun 2012 **Visiting Research Student at CSAIL**, *Massachusetts Institute of Technology (MIT)*, Cambridge, MA, USA.

Jan–Jun 2009 **Erasmus Exchange Student**, *Chalmers University of Technology*, Gothenburg, Sweden.

Theses

Master Thesis (Jan – Jun 2012)

Title **Mapping of Unknown Space Targets for Relative Navigation and Inspection**
School *CSAIL, Massachusetts Institute of Technologie (MIT), Cambridge, MA, USA*
Advisor Prof. Dr. John J. Leonard
Description Implementation of visual SLAM algorithms.
Successfully tested aboard the International Space Station (ISS).
Grade 6.0/6.0

Semester Thesis (Feb – Jul 2011)

Title **Robotic calligraphy – A robot that learns how to write Chinese calligraphy**
School *Institute for Dynamic Systems and Control, ETH, Zurich, Switzerland*
Advisor Prof. Dr. Raffaello D'Andrea

Description Control of a robotic manipulator with 7 degrees of freedom (KUKA LWR).
Implementation of an iterative learning controller.

Grade 6.0/6.0

Bachelor Thesis (Jan – Jun 2009)

Title **Elimination of Slugs – Defending a Security Line against Slugs**

School *Institution for Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden*

Advisor Prof. Dr. Jonas Fredriksson

Description Mechanical and electrical design and prototype building.
Controller implementation on microprocessor.
Implementation of a visual slug detector.

Grade 6.0/6.0

Matriculation Thesis (2004 – 2006)

Title **Realisation of a Book Scanner, and Programming of an OCR Software**

School *High School, Kreuzlingen, Switzerland*

Advisor Guido Lang

Description Designing and building a working prototype.
Programming pneumatic logic on PLC (Siemens S7-200).
Own implementation of an optical character recognition system (OCR).
Winner of Swiss Contest for Young Scientists 2006, Basel, Switzerland.
Qualification for EU Contest for Young Scientists 2006, Stockholm, Sweden.

Grade 6.0/6.0

Teaching Experience

2015 **Teaching Assistant for Computer Vision**, *University of Zurich*, Zurich, Switzerland.
Exercise sessions
Supervisor: Prof. Dr. Davide Scaramuzza

2013 **Teaching Internship for Control Theory**, *Zurich University of Applied Science (ZHAW)*, Winterthur, Switzerland, *Grade 5.75/6.0*.
Classes and laboratory sessions (20 lessons total)
Supervisor: Prof. Dr. Roland Büchi

2010 **Teaching Assistant for Calculus**, *ETH*, Zurich, Switzerland.
Preparation and supervision of exercises.
Supervisor: Prof. Dr. Marc Burger

2008 **Teaching Assistant for Mechanics**, *Center of Mechanics, ETH*, Zurich, Switzerland.
Preparation and supervision of exercises and exams (plus grading).
Supervisor: Prof. Dr. Jürg Dual

Professional Experience

Jan–Jun 2010 **Industrial Internship**, *MTU Aero Engines GmbH*, Munich, Germany.
Pre-design tool to analyse multi-dody damping in blades and vane clusters.

2007 – 2008 **Side Job**, *energieburo.ch*, Zurich, Switzerland.
Creating photo-realistic 3D-CAD models of solar panels.

Feb–Mar 2007 **Workshop Internship**, *MOWAG GmbH*, Kreuzlingen, Switzerland.
Practical training on drilling, milling, lathing, and welding.

2006 **Web Developer**, *High School Burggraben*, St. Gallen, Switzerland.
Development of an online room reservation system.
200+ users, 100+ rooms, 1M+ reservations. Still in use (2017).

Awards

2017 Misha Mahowald Prize for Neuromorphic Engineering

2016 NCCR Robotics PhD/Postdoc Exchange Grant

2016 IROS 16 Best Application Paper Award Finalist

2016 Qualcomm Innovation Fellowship Europe ($40.000)

2014 SSRR 14 Best Paper Award Finalist

2014 Winner of KUKA Innovation Award (€20.000)

2014 Convergent Science Network of Biomimetics and Neurotechnology CapoCaccia Fellowship

2012 Hans und Wilma Stutz Foundation Scholarship

2006 Winner of Swiss Contest for Young Scientists

2006 Technorama Prize for Young Scientists

2006 Metrohm Foundation Prize for Young Scientists

2004 1st Place in Category at ThinkQuest Swiss Web Award

Other Activities

2014 – 2015 **Ph.D. Representative**, *Department of Informatics, University of Zurich*, Zurich, Switzerland.

2003 – 2011 **Scouts Leader**, *Jungwacht Weinfelden*, Switzerland.
Head of Organization (2008 – 2009)
Organization has 25 leaders and 80 kids.

2005 – 2006 **President of Student Organization**, *High School*, Kreuzlingen, Switzerland.

Public Exhibitions

Sep 2015 **Scientifica, Zurich, Switzerland** *25.000 visitors*

May 2015 **SwissCore 20th Anniversary, Brussels, Belgium**.
Research demonstration to EU and SNSF officials, including Jean-Pierre Bourguignon (ERC president), Martin Vetterli (President of the Research Council of the SNSF) and Michael Hengartner (President of the University of Zurich)

Jun 2014 **AUTOMATICA, Munich, Germany** *31.000 visitors*

Sep 2013	**tunZurich, Zurich, Switzerland**	*2.500 visitors*
Aug 2013	**Scientifica, Zurich, Switzerland**	*20.000 visitors*
Apr 2013	**Festival de Robotique, Lausanne, Switzerland**	*20.000 visitors*
Mar 2013	**Robots on Tour, Zurich, Switzerland**	*4.000 visitors*

Talks

Mar 28, 2017 **Robotics and Interactions (RIS) Group Seminar**, *LAAS-CNRS Toulouse*.
"Towards Agile Flight of Vision-controlled Drones" (with Davide Falanga)

Mar 15, 2017 **Volkshochschule Winterthur**, *ZHAW Winterthur, Switzerland*.
"Autonom fliegende Roboter" (in German)

Nov 17, 2016 **EMBE HSG Alumni**, *Technopark Zurich, Switzerland*.
"Autonome Flugroboter" (in German)

Oct 25, 2016 **Naturwissenschaftliche Gesellschaft Thurgau**, *Kreuzlingen, Switzerland*.
"Autonome Flugroboter auf Rettungsmission" (in German)

Jun 9, 2016 **Kiwanis**, *Küsnacht, Switzerland*.
"Autonom fliegende Quadrokopter" (in German)

May 10, 2016 **Qualcomm Innovation Fellowship Finals**, *University of Amsterdam*.
"Event-based Vision for High-Speed Robotics"

Sep 8, 2015 **"Technologieoutlook und IT-Trends als Chance für Europa — Digital Society and Economy 4.0"**, *University of Zurich*.
"Autonom fliegende Quadrokopter" (in German)

Aug 12, 2015 **Science Week 2015**, *High School Kreuzlingen*.
"Autonom fliegende Quadrokopter" (in German)

Oct 27, 2014 **Workshop on "Obstacles on the way to a broad deployment of robot technology in disaster response"**, *SSRR 2014*.
"NCCR Robotics: Switzerland's Rescue Robotics Grand Challenge"

Sep 18, 2014 **3rd Workshop on Visual Control of Mobile Robots**, *IROS 2014*.
"Appearance-based, active vision applied to autonomous mapping from micro aerial vehicles"

Sep 15, 2014 **Aerial Open Source Robotics Workshop**, *IROS 2014*.
"Open-Source Packages for Real-Time Pose Estimation of Micro Aerial Vehicles"

Jun 27, 2014 **Institute for Computer Science and Control, Hungarian Academy of Sciences, Budapest, Hungary**, *Dr. Andras Majdik*.
"Event-based, 6-DOF Pose Tracking for High-Speed Maneuvers"

Aug 1, 2012 **Autonomous Intelligent Systems, University of Freiburg, Germany**, *Prof. Dr. Wolfram Burgard*.
"Visual Mapping of Unknown Space Targets for Relative Navigation and Inspection"

Jul 19, 2012 **fortiss, TU Munich, Germany**, *Dr. Markus Rickert*.
"Visual Mapping of Unknown Space Targets for Relative Navigation and Inspection"

Jul 16, 2012 **Institute for Computer Graphics and Vision, TU Graz, Austria**, *Prof. Dr. Horst Bischof*.
"Visual Mapping of Unknown Space Targets for Relative Navigation and Inspection"

Attended Summer Schools

May 2014 **CapoCaccia Cognitive Neuromorphic Engineering Workshop**, *Alghero, Italy*.

Oct 2013 **Rescue Robotics Camp**, *Linkoping, Sweden*.

Jul 2013 **International Computer Vision Summer School (ICVSS)**, *Calabria, Italy*.

Attended Conferences

Jun 2016 **International Conference on Event-Based Control, Communication and Signal Processing (EBCCSP)**, *Krakow, Poland*.

Sep 2015 **European Conference on Mobile Robots (ECMR)**, *Lincoln, United Kingdom*.

July 2015 **Robotics: Science and Systems (RSS)**, *Rome, Italy*.

May 2015 **IEEE International Conference on Robotics and Automation (ICRA)**, *Seattle WA, USA*.

Oct 2014 **IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)**, *Toyako-cho, Japan*.

Sep 2014 **IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**, *Chicago IL, USA*.

Sep 2014 **European Conference on Computer Vision (ECCV)**, *Zurich, Switzerland*.

May 2013 **IEEE International Conference on Robotics and Automation (ICRA)**, *Karlsruhe, Germany*.

Jun 2013 **International Workshop on the Algorithmic Foundations of Robotics (WAFR)**, *Cambridge MA, USA*.

Students

2016 **Timo Horstschäfer**, *ETH Zurich*, Master Thesis, **awarded Fritz Kutter Award 2016**.
"Parallel Tracking, Depth Estimation, and Image Reconstruction with an Event Camera"

2016 **Jonathan Huber**, *ETH Zurich*, Semester Thesis.
"Ground Robot Localization in Aerial 3D Maps"

2016 **Julia Nitsch**, *University of Graz*, NCCR Internship.
"Terrain Classification in Search-and-Rescue Scenarios"

2016 **Beat Küng**, *ETH Zurich*, Master Thesis.
"Visual Odometry pipeline for the DAVIS camera"

2015 **Mathis Kappeler**, *University of Zurich*, Master Project.
"Exposure Control for Robust Visual Odometry"

2015 **Imanol Studer**, *University of Zurich*, Master Project.
"Head Pose Tracking with Quadrotors"

2015 **Jon Lund**, *University of Zurich*, Master Thesis.
"Towards SLAM for Dynamic Vision Sensors"

2015 **Micha Brunner**, *ETH Zurich*, Semester Thesis.
"Flying Motion Capture System"

2015 **Igor Bozic**, *University of Zurich*, Master Project.
"High-Frequency Position Control of the KUKA youBot Arm"

2015 **Joachim Ott**, *ETH Zurich*, Semester Thesis.
"Vision-Based Surface Classification for Micro Aerial Vehicles"

2015 **David Tedaldi**, *ETH Zurich*, Semester Thesis.
"Feature Tracking based on Frames and Events"

2015 **Nathan Baumli**, *ETH Zurich*, Master Thesis.
"Towards Evasive Maneuvers for Quadrotors using Stereo Dynamic Vision"

2014 **Amos Zweig**, *ETH Zurich*, Semester Thesis.
"Event-based Depth Estimation"

2014 **Nathan Baumli**, *ETH Zurich*, Semester Thesis.
"Event-Based Full-Frame Visualization"

2014 **Basil Huber**, *EPFL Lausanne*, Master Thesis, **awarded Fritz Kutter Award 2014**.
"High-Speed Pose Estimation using a Dynamic Vision Sensor"

2013 **Karl Schwabe**, *ETH Zurich*, Master Thesis.
"A Monocular Pose Estimation System based on Infrared LEDs"

2013 **Benjamin Keiser**, *ETH Zurich*, Master Thesis, **awarded KUKA Best Student Project Award 2013**.
"Torque Control of a KUKA youBot Arm"

Service

Journal Reviewer

TPAMI	**IEEE Transactions on Pattern Analysis and Machine Intelligence**	*2017*
JFR	**Journal of Field Robotics**	*2017*
TNNLS	**IEEE Transactions on Neural Networks and Learning Systems**	*2015*

Conference Reviewer

IROS	**IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems**	*2014–2016*
ICRA	**IEEE Intl. Conf. on Robotics and Automation**	*2014–2017*
EBCCSP	**IEEE Intl. Conf. on Event-Based Control Comm. Signal Process.**	*2015*
ISCAS	**IEEE Intl. Symp. on Circuits and Systems**	*2016*
ISER	**Intl. Symp. on Experimental Robotics**	*2016*
SSRR	**IEEE Intl. Symp. on Safety, Security, and Rescue Robotics**	*2016*

Student Volunteer

	IFI Summer School	*2015, 2017*
ECCV	**European Conference on Computer Vision**	*2014*

Committee

bugnplay.ch	**Swiss Youth Contest for Media and Robotics**	*2015, 2016*

Journal Articles

[1] G. Gallego, J. E. Lund, **E. Mueggler**, H. Rebecq, T. Delbruck, and D. Scaramuzza, "Event-based, 6-DOF camera tracking for high-speed applications," *IEEE Trans. Pattern Anal. Machine Intell.*, 2017.

[2] H. Rebecq, G. Gallego, **E. Mueggler**, and D. Scaramuzza, "EMVS: Event-based multi-view stereo: 3D reconstruction with an event camera," *International Journal of Computer Vision*, 2017, under review.

[3] **E. Mueggler**, G. Gallego, H. Rebecq, and D. Scaramuzza, "Continuous-time visual-inertial trajectory estimation with event cameras," *IEEE Trans. Robotics*, 2017, under review.

[4] **E. Mueggler**, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM," *Intl. J. of Robotics Research*, vol. 36, pp. 142–149, 2 2017.

[5] J. Delmerico, **E. Mueggler**, J. Nitsch, and D. Scaramuzza, "Active autonomous aerial exploration for ground robot path planning," *IEEE Robotics and Automation Letters*, vol. 2, pp. 664–671, 2 2017.

[6] M. Faessler, F. Fontana, C. Forster, **E. Mueggler**, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3D mapping with a quadrotor micro aerial vehicle," *J. of Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016, ISSN: 1556-4967.

[7] C. Graber and **E. Müggler**, "Vom Buch zur Textdatei. Bau und Programmierung einer Bucheinscannmaschine," *Junge Wissenschaft*, vol. 76, pp. 10–15, 2007, in German.

Peer-Reviewed Conference Papers

[1] **E. Mueggler**, C. Bartolozzi, and D. Scaramuzza, "Fast event-based corner detection," in *British Machine Vision Conf. (BMVC)*, under review, 2017.

[2] V. Vasco, A. Glover, **E. Mueggler**, D. Scaramuzza, L. Natale, and C. Bartolozzi, "Independent motion detection with event-driven cameras," in *IEEE Intl. Conf. on Advanced Robotics (ICAR)*, Hong Kong, China, Jul. 2017.

[3] D. Falanga, **E. Mueggler**, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Singapore, May 2017.

[4] R. Kaeslin, P. Fankhauser, E. Stumm, Z. Taylor, **E. Mueggler**, J. Delmerico, D. Scaramuzza, R. Siegwart, and M. Hutter, "Collaborative localization of aerial and ground robots through elevation maps," in *IEEE Intl. Symp. on Safety, Security, and Rescue Robotics (SSRR)*, Lausanne, Switzerland, Oct. 2016.

[5] B. Kueng, **E. Mueggler**, G. Gallego, and D. Scaramuzza, "Low-latency visual odometry using event-based feature tracks," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Daejeon, Korea, Oct. 2016.

[6] D. Tedaldi, G. Gallego, **E. Mueggler**, and D. Scaramuzza, "Feature detection and tracking with the dynamic and active-pixel vision sensor (DAVIS)," in *Intl. Conf. on Event-Based Control, Comm. and Signal Proc. (EBCCSP)*, Krakow, Poland, Jun. 2016.

[7] J. Delmerico, **E. Mueggler**, A. Giusti, and D. Scaramuzza, "'On-the-spot training' for terrain classification in autonomous air-ground collaborative teams," in *Intl. Sym. on Experimental Robotics (ISER)*, Tokyo, Japan, Oct. 2016.

[8] **E. Mueggler**, N. Baumli, F. Fontana, and D. Scaramuzza, "Towards evasive maneuvers with quadrotors using dynamic vision sensors," in *Eur. Conf. on Mobile Robots (ECMR)*, Lincoln, England, Sep. 2015.

[9] **E. Mueggler**, G. Gallego, and D. Scaramuzza, "Continuous-time trajectory estimation for event-based vision sensors," in *Robotics: Science and Systems (RSS)*, Rome, Italy, Jul. 2015.

[10] **E. Mueggler**, C. Forster, N. Baumli, G. Gallego, and D. Scaramuzza, "Lifetime estimation of events from dynamic vision sensors," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Seattle, WA, May 2015.

[11] T. Delbruck, M. Pfeiffer, R. Juston, G. Orchard, **E. Müggler**, A. Linares-Barranco, and M. W. Tilden, "Human vs. computer slot car racing using an event and frame-based DAVIS vision sensor," in *IEEE Intl. Symp. on Circuits and Systems (ISCAS)*, Lisbon, Portugal, May 2015.

[12] **E. Mueggler**, M. Faessler, F. Fontana, and D. Scaramuzza, "Aerial-guided navigation of a ground robot among movable obstacles," in *IEEE Intl. Symp. on Safety, Security, and Rescue Robotics (SSRR)*, Toyako-cho, Japan, Oct. 2014.

[13] **E. Mueggler**, B. Huber, and D. Scaramuzza, "Event-based, 6-DOF pose tracking for high-speed maneuvers," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Chicago, IL, Sep. 2014.

[14] M. Faessler, **E. Mueggler**, K. Schwabe, and D. Scaramuzza, "A monocular pose estimation system based on infrared LEDs," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Hong Kong, China, May 2014.

[15] N. Huebel, **E. Mueggler**, M. Waibel, and R. D'Andrea, "Towards robotic calligraphy," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, Oct. 2012.

[16] B. E. Tweddle, E. Müggler, A. Saenz-Otero, and D. W. Miller, "The SPHERES VERTIGO goggles: Vision based mapping and localization onboard the International Space Station," in *Intl. Symp. on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*, Turin, Italy, Sep. 2012.

Other Publications

[1] M. Faessler, **E. Mueggler**, and D. Scaramuzza, *Happy Easter from the AI Lab*, Video Competition at International Joint Conferences on Artificial Intelligence (IJCAI), Beijing, China, Aug. 2013.

[2] M. F. Fallon, H. Johannsson, M. Kaess, D. M. Rosen, **E. Muggler**, and J. J. Leonard, *Mapping the MIT Stata Center: Large-scale integrated visual and RGB-D SLAM*, RGB-D Workshop at Robotics: Science and Systems Conference (RSS), Sydney, Australia, Jul. 2012.