

Event-Based Computer Vision for Fast Robot Control

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Andreas Alfred Ziegler
aus Zürich/Schweiz

Tübingen
2025

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen
Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	23.01.2026
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. rer. nat. Andreas Zell
2. Berichterstatter:	Prof. Dr.-Ing. Andreas Geiger

Abstract

Table tennis is a fast-paced sport requiring precise perception and agility to win a game. This makes table tennis an ideal test bed for robotic research. Thus, it is no surprise that in the past years, more research groups have started developing perception and control algorithms for table tennis playing robots. So far, all the table tennis robots in the literature use frame-based cameras for their perception pipeline. Given decades of research, mature algorithms for frame-based cameras are available. However, frame-based cameras face a limit in very dynamic scenes. The motion blur present in frames can lead to decreased accuracy or failure of frame-based pipelines. One way to counteract this is to reduce the exposure time and use frame-based cameras with a high frame rate. While this is fine from the sensor's perspective, it shifts the bottleneck to the algorithm, which now has to process more data at the same time. Another approach is using event-based cameras. These novel sensors report changes in illumination intensity with so-called events for every pixel independently and asynchronously. This leads to an event stream with low latency, a temporal resolution of μs , a high dynamic range, and data sparsity. Event-based cameras are thus an interesting sensor for table tennis robots with the potential to overcome the shortfalls of frame-based cameras.

However, event-based cameras output a stream of events, each containing the pixel's position, a time stamp, and the polarity of the intensity change. Therefore, the output of an event-based camera can not be processed by existing computer vision pipelines that use frames as input. In this PhD thesis, *Event-Based Computer Vision for Fast Robot Control*, we analyzed, developed, and evaluated event-based computer vision algorithms for the perception pipeline of a table tennis robot. Our contribu-

tions show that event-based computer vision can overcome the limitations of frame-based cameras. We show that an event-based ball detector can achieve an update rate more than an order of magnitude higher than a frame-based pipeline. This can help reduce the uncertainties of the predicted trajectory and the ball's spin, which is beneficial for robot control. The event-based spin estimation algorithm developed here delivers improved accuracy for fast-spinning balls, making it particularly valuable for challenging table tennis rallies where precise spin estimates are crucial.

In addition to these more table tennis-related contributions to perception, we also developed algorithms that are prerequisites for deploying event-based cameras in multi-view scenarios and in combination with frame-based cameras. These contributions are a calibration framework for wide-baseline camera systems with frame-based and event-based cameras and a benchmark for tuning the settings of an event-based camera.

Kurzfassung

Tischtennis ist ein dynamischer Sport, der präzise Wahrnehmung und hohe Agilität erfordert, um ein Spiel zu gewinnen. Diese Eigenschaften machen Tischtennis zu einer idealen Testumgebung für die Robotikforschung. Es ist daher wenig überraschend, dass in den vergangenen Jahren vermehrt Forschungsgruppen Wahrnehmungs- und Steuerungsalgorithmen für tischtennisspielende Roboter entwickelt haben. Bisher setzen alle in der Literatur beschriebenen Tischtennisroboter auf bildbasierte Kameras innerhalb ihrer Wahrnehmungspipeline. Aufgrund jahrzehntelanger Forschung stehen hierfür ausgereifte Algorithmen zur Verfügung. Allerdings stoßen bildbasierte Kameras in hochdynamischen Szenen an ihre Grenzen: Bewegungsunschärfe kann die Genauigkeit der Bildauswertung erheblich verringern oder sogar zum Versagen der gesamten Pipeline führen. Eine Möglichkeit zur Minderung dieser Problematik besteht darin, die Belichtungszeit zu verkürzen und Kameras mit hoher Bildfrequenz einzusetzen. Während dies aus Sensortechnischer Sicht vorteilhaft ist, verschiebt sich die Hauptbelastung auf die Algorithmen, die eine deutlich größere Datenmenge in kürzerer Zeit verarbeiten müssen. Eine alternative Lösung stellt der Einsatz von Event-basierten Kameras dar. Diese neuartigen Sensoren detektieren Änderungen der Beleuchtungsintensität unabhängig und asynchron für jedes Pixel einzeln. Daraus resultiert ein Event-Stream mit geringer Latenz, einer zeitlichen Auflösung im Mikrosekundenbereich, einem hohen Dynamikbereich sowie einer ausgeprägten Datensparsamkeit. Event-basierte Kameras sind daher ein vielversprechender Sensortyp für Tischtennisroboter und bieten das Potenzial, die Limitierungen bildbasierter Kameras zu überwinden.

Allerdings liefern Event-basierte Kameras keine konventionellen Bilder, sondern einen kontinuierlichen Stream einzelner Events, bestehend

aus Pixelposition, Zeitstempel und Polarität der Helligkeitsänderung. Daher können bestehende Verfahren der Bildverarbeitung, die auf der Verarbeitung von Einzelbildern basieren, nicht direkt angewendet werden. In dieser Dissertation, *Event-Based Computer Vision for Fast Robot Control*, wurden Event-basierte Algorithmen für die Wahrnehmungspipeline eines Tischtennisroboters analysiert, entwickelt und evaluiert. Die Beiträge dieser Arbeit zeigen, dass ereignisbasierte Bildverarbeitung die Grenzen bildbasierter Ansätze überwinden kann. Es wird demonstriert, dass ein Event-basierter Balldetektor eine Aktualisierungsrate erreicht, die mehr als eine Größenordnung über der bildbasierter Pipelines liegt. Dadurch können Unsicherheiten in der Vorhersage der Balltrajektorie sowie des Ballspins reduziert werden, was die Leistungsfähigkeit der Robotersteuerung erheblich verbessert. Der im Rahmen dieser Arbeit entwickelte Algorithmus zur Spin-Schätzung auf Basis Event-basierten Daten ermöglicht zudem eine erhöhte Genauigkeit insbesondere bei schnell rotierenden Bällen und ist somit von besonderem Wert für anspruchsvolle Spielsituationen im Tischtennis, in denen präzise Spin-Schätzungen entscheidend sind.

Neben diesen spezifischen Beiträgen im Bereich der Wahrnehmung für Tischtennisroboter wurden in dieser Arbeit auch grundlegende Algorithmen entwickelt, die den Einsatz Event-basierten Kameras in Multikamera-Szenarien sowie in Kombination mit bildbasierten Kameras ermöglichen. Dazu zählen ein Kalibrierungsframework für Kamerasysteme mit großer Basislinie, bestehend aus bildbasierten und ereignisbasierten Kameras, sowie ein Benchmark zur Optimierung der Sensoreinstellungen ereignisbasierter Kameras.

Acknowledgments

I want to express my gratitude to Peter Dürr from Sony AI, who made this PhD project possible. During my time at Sony AI, Peter has also shown me a positive example of what modern leadership can look like. Next, I want to thank Andreas Zell for providing me with the infrastructure to conduct my PhD and for giving me the freedom to explore different research topics and collaborations. The Cognitive Systems Group would be a pretty sterile place without the presence and deeds of Uli Ulmer. Uli allowed me to use Arch Linux on my PC and found a pragmatic solution for almost all technical problems we faced. He also demonstrated a great level of flexibility, particularly when we approached him with numerous last-minute requests. Next to all this, he was also the person who cheered up the whole group, e.g., by bringing oranges, cakes, taking care of our plants on the balcony, organizing barbecues, and many more things. Another person who made my life in the lab simpler is Vita Serbakova. She spotted the countless mistakes I had in the official forms of the University and always made sure we didn't miss out.

My PhD journey would have been half as fruitful and certainly more painful without my colleagues. Firstly, I want to express my gratitude to Jonas Tebbe, Thomas Gossard, and David Joseph. Jonas helped me navigate the lab and academia when I started my PhD. Throughout my PhD, I worked closely with Thomas. Not only do I enjoy his rigorous scientific mindset and his proactive and pragmatic problem-solving skills, but we also shared a fair amount of laughter, early-morning paper submissions, frustration, discussions, and a hotel room in Yokohama. While I first met David as a Bachelor's student doing his thesis under my supervision, I enjoyed the time we shared in the lab as colleagues. David helped me when my physics knowledge hit a wall or when I needed a new and fresh

perspective, and he was also my personal driver when needed.

Other colleagues at the Cognitive Systems Group I want to mention are: Martin Messmer who inspired me with his sustainable life style; Benjamin Kiefer who was an expert in the unwritten rules in the lab; Maximus Mutschler who inspired me with his activism and showed me how minimalism can be applied to cooking; Yitong Quan who managed to always surprise me with his unconventional ideas and ways of cooking; Marcel Hallgarten who encouraged me to contact other researchers; Mario Laux who inspired me to explain complex topic with simple language; Max Beffert who provided me the consistency in my life with his lunch selection; Further colleagues I appreciated having conversation with are Rafia Rahim, David Ott, Dominik Hildebrand, Boya Zhang.

In addition to my colleagues, I also had a number of Bachelor's and Master's students I could supervise. It was a formative experience not only supervising students but also learning from them, their ideas, and seeing them grow.

I would have had a hard time in my PhD without the mentors who guided me. Starting from a GitHub issue I created, followed by some emails, Bernd Pfrommer is not only the author of many tools for event cameras I used, but he also proved himself to be an invaluable help when I was stuck in the details of events. I started working with Sebastian Otte when we co-supervised a Master's thesis together. Later on, he helped me to improve my paper writing skills and was always available when I had questions, concerns, or worries during my PhD. Sebastian's transparent and authentic way was also an inspiration for my own leadership style. I contacted Arren Glover with some questions about a previous publication. From there, we had many discussions and collaborated on one of my projects. Arren guided me on experiment design, writing, and strategies in academia, and was also a great help in my first workshop proposal for IROS 2025. I met Giulia D'Angelo at a neuromorphic hackathon at ZHAW. From there, we started collaborating on an ongoing project. Next to this, Giulia also helped me navigate the mental load of a PhD.

During my PhD, I also struggled with some personal challenges. I'm very grateful for the support and strength Friends from the Religious So-

ciety of Friends (Quakers) gave me. Furthermore, Friends helped and encouraged me to ask fundamental and sometimes difficult questions in life, in addition to those in research. I'm also grateful for the trust Friends gave me in my capacity as a co-clerk of the Switzerland Yearly Meeting, in which capacity I could grow as a facilitator and leader. Next to the stillness in the Quaker meetings for worship, I also intensified my Zen practice. I want to thank my Zen Sangha for giving me the time and space to get closer to my true self, whatever this might be.

Throughout my PhD, alongside my research and spiritual journey, I had a couple of friends who supported me when I needed it. My recurrent mastermind sessions with Philipp Riedel are indispensable. Philipp has a similarly broad interest, which makes me always a bit smarter after a conversation with him. His nonjudgmental, scientific, analytical, yet emphatic way makes him a great friend, supporter, and also sparring partner for ideas. While Simeon Schaub was first "just" a Friend, we also got in touch outside of the Quaker bubble. Simeon is a great help, shows me new perspectives in life and spirituality, and is a great companion. I'm a big mountain enthusiast and happy that I can share this passion and challenge with Sandro Bergamin, Marco Bergamin, Philippe Tschanz, Tobias Krieg, and my brother Thomas Ziegler. Next to ski-touring, mountaineering, climbing, and hiking, I also enjoyed the many discussions, concerts, and beers with you guys.

A PhD can be time-demanding; mine was. I'm very grateful for my parents, Fredi and Ursula Ziegler, for their unconditional support throughout my life. Although they are not familiar with academia, they made an effort to understand my challenges. Last but not least, I want to thank my brother Thomas Ziegler. Not only did he help me with my research, e.g., helping me decide which conference to submit a paper to, discussing research topics, C++ questions, or all the last-day reviews he gave me, Thomas was also my most loyal ski-touring and mountaineering companion.

List of Contributions

Journal Publications

- Ziegler, A., Gossard, T., Glover, A., and Zell, A. (2025c). An event-based perception pipeline for a table tennis robot. *IEEE Robotics and Automation Letters*. to be submitted

The whole pipeline and all the experiments were done by me. I also wrote 90% of the manuscript. Thomas Gossard helped me with the hardware setup and the calibration of the camera system, contributed around 5% to the manuscript, and gave me feedback and insights. Arren Glover partially supervised me on this project. He also contributed around 5% to the manuscript. Andreas Zell reviewed the manuscript and gave me feedback. I consider my contribution to this work to be 90% of the total work.

Links: Appendix F, PDF

Peer-Reviewed Conference Papers

- Ziegler, A., Vetter, K., Gossard, T., Tebbe, J., Otte, S., and Zell, A. (2025b). Detection of fast-moving objects with neuromorphic hardware. In *2025 International Conference on Robotics and Automation (ICRA)*. IEEE

I developed the software to integrate the Spiking Neural Network (SNN) into the robot system for the real-time experiment, and conducted the real-time experiment. I also wrote 80% of the manuscript, managed the project, and su-

pervised Karl Vetter during this project. Karl Vetter developed all the SNN networks and conducted all the offline experiments. He also contributed 10% to the manuscript. Sebastian Otte contributed 10% to the manuscript and partially supervised me on this project. Thomas Gossard helped us with the robot setup, reviewed the manuscript, and gave feedback. Jonas Tebbe also helped us with the robot setup, reviewed the manuscript, and gave feedback. Andreas Zell reviewed the manuscript and gave me feedback. I consider my contribution to this work to be 60% of the total work.

Links: Appendix D, PDF, Project, Video

- Gossard, T., Ziegler, A., Kolmar, L., Tebbe, J., and Zell, A. (2024a). ewand: A calibration framework for wide baseline frame-based and event-based camera systems. In *2024 International Conference on Robotics and Automation (ICRA)*. IEEE

I wrote 80% of the bundle adjustment code, developed the event-based marker detection system, contributed 50% to all the experiments, and wrote 50% of the manuscript. Thomas Gossard made the hardware design, contributed 20% to the bundle adjustment code, developed the frame-based marker detection system, contributed 50% to all the experiments, and wrote 50% of the manuscript. Levin Kolmar did his Bachelor's thesis on the topic under the supervision of Thomas Gossard and me. Parts of his work helped us to improve the project. Jonas Tebbe helped us with the camera system, reviewed the manuscript, and gave feedback. Andreas Zell reviewed the manuscript and gave us feedback. I consider my contribution to this work to be 45% of the total work.

Links: Appendix B, PDF, Project, Video

- Gossard, T., Tebbe, J., Ziegler, A., and Zell, A. (2023). Spindoe: A ball spin estimation method for table tennis robot. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

IEEE

Thomas Gossard had the idea, wrote 90% of the pipeline, conducted all the experiments, and wrote the manuscript. Jonas Tebbe wrote 10% of the pipeline, contributed to the idea, helped with the experiments, reviewed the manuscript, and gave feedback. I gave feedback during the project, helped with the camera system, reviewed the manuscript, and gave insightful feedback. Andreas Zell reviewed the manuscript and gave us feedback. I consider my contribution to this work to be 5% of the total work.

Links: PDF, Project

- Ziegler, A., Teigland, D., Tebbe, J., Gossard, T., and Zell, A. (2023b). Real-time event simulation with frame-based cameras. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE

I implemented the whole software using code from a previous Bachelor's thesis by Daniel Teigland. I conducted 80% of all the experiments and wrote the entire manuscript. Daniel Teigland did his Bachelor thesis on this topic under the supervision of me and Jonas Tebbe, contributed to the software implementation, and conducted 20% of the experiments. Jonas Tebbe contributed to the idea, reviewed the manuscript, and gave feedback. Thomas Gossard reviewed the manuscript and gave feedback. Andreas Zell contributed to the initial idea, reviewed the manuscript, and gave us feedback. I consider my contribution to this work to be 70% of the total work.

Links: Appendix A, PDF, Poster, Project, Video

Peer-Reviewed Conference Workshop Papers

- Gossard, T., Schmalzl, J., Ziegler, A., and Zell, A. (2025a). Spin detection using racket bounce sounds in table tennis. In *2025 IEEE*

International Workshop on Sport, Technology and Research (STAR).
IEEE

Thomas Gossard had the idea, wrote the majority of the pipeline, conducted the majority of the experiments, and wrote the manuscript. Julian Schmalzl contributed to the pipeline and conducted some experiments. I gave feedback during the project and reviewed the manuscript. Andreas Zell reviewed the manuscript and gave us feedback. I consider my contribution to this work to be 5% of the total work.

Links: PDF

- Ziegler, A., Joseph, D., Gossard, T., Moldovan, E., and Zell, A. (2025a). Biasbench: A reproducible benchmark for tuning the biases of event cameras. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE

I was in charge of the project, contributed ideas, set up the Visual Odometry (VO) pipeline, ran the VO pipeline on all the recorded trajectories, developed a time synchronization tool to compare the trajectories, and evaluated the trajectories. I also wrote 80% of the manuscript and supervised two students (David Joseph and Emil Moldovan) who both did their Bachelor's theses on this topic. Thomas Gossard contributed ideas, co-developed the blinking LED panel, built the robot arm setup, recorded trajectories, and performed the ball tracking experiment. He also wrote 15% of the manuscript and helped supervise the two Bachelor students who worked on the project. David Joseph developed the Behavior Cloning pipeline, was in charge of most of the recordings, and performed the experiment with the LED panel. He also co-developed the blinking LED panel and wrote 5% of the manuscript. Before starting as a research assistant, David Joseph conducted his Bachelor's thesis with Emil Moldovan under my supervision. Andreas Zell re-

viewed the manuscript and gave us feedback. I consider my contribution to this work to be 33% of the total work.

Links: Appendix C, PDF, Project, Video

- Gossard, T., Ziegler, A., , and Zell, A. (2025b). Tt3d: Table tennis 3d reconstruction. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE

Thomas Gossard had the idea, wrote the pipeline, conducted all the experiments, and wrote the manuscript. I gave feedback during the project and reviewed the manuscript. Andreas Zell reviewed the manuscript and gave us feedback. I consider my contribution to this work to be 5% of the total work.

Links: PDF, Project

- Gossard, T., Krismer, J., Ziegler, A., Tebbe, J., and Zell, A. (2024b). Table tennis ball spin estimation with an event camera. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, page 3347–3356. IEEE

Thomas Gossard contributed to the idea, implemented 40% of the whole pipeline, contributed 50% of the experiments, wrote 20% of the manuscript, and co-supervised Julian Krismer. Julian Krismer did his Master’s thesis under the supervision of Thomas Gossard and me. He contributed to the idea, implemented 60% of the whole pipeline, contributed 50% of the experiments, and wrote 20% of the manuscript. I contributed to the idea, wrote 60% of the manuscript, and co-supervised Julian Krismer. I consider my contribution to this work to be 20% of the total work.

Links: Appendix E, PDF

- Ziegler, A., Gossard, T., Vetter, K., Tebbe, J., and Zell, A. (2023a). A multi-modal table tennis robot system. In *RoboLetics: Workshop on Robot Learning in Athletics @ CoRL 2023*

I contributed to the ideas, conducted 30% of the experiments, and wrote 60% of the manuscript. Thomas Gossard contributed to the ideas, conducted 30% of the experiments, and wrote 40% of the manuscript. Karl Vetter did his Master's thesis under my supervision, conducted 30% of the experiments, reviewed the manuscript, and gave us feedback. Andreas Zell contributed to the initial idea, reviewed the manuscript, and gave us feedback. I consider my contribution to this work to be 50% of the total work.

Links: PDF

Contents

List of Contributions	xi
1 Introduction	1
1.1 Motivation	2
1.1.1 A table tennis robot system	3
1.1.2 Event-based computer vision	4
1.1.3 Event-based computer vision for fast robot control	4
1.2 Related Work	5
1.2.1 Event-based camera simulation	6
1.2.2 Event-based camera calibration	7
1.2.3 Bias tuning for event cameras	9
1.2.4 Ball detection	11
1.2.5 Spin estimation	12
1.3 Problem Statement	13
1.3.1 Event-based camera simulation	13
1.3.2 Camera calibration	13
1.3.3 Automatic bias tuning	13
1.3.4 Ball detection	14
1.3.5 Spin estimation	14
2 Contributions	15
2.1 Event-based camera simulation	15
2.1.1 Paper Appendix A: Real-time event simulation with frame-based cameras	16

2.2	Camera calibration	21
2.2.1	Paper Appendix B: eWand: A calibration framework for wide baseline frame-based and event-based camera systems	21
2.3	Automatic bias tuning	23
2.3.1	Paper Appendix C: BiasBench: A reproducible benchmark for tuning the biases of event cameras	24
2.4	Event-based ball detection	24
2.4.1	Paper Appendix F: An Event-Based Perception Pipeline for a Table Tennis Robot	25
2.4.2	Paper Appendix D: Detection of Fast-Moving Objects with Neuromorphic Hardware	27
2.5	Event-based spin estimation	29
2.5.1	Paper Appendix E: Table tennis ball spin estimation with an event camera	29
2.6	Unrelated contributions	31
3	Conclusion	33
3.1	Conclusion	33
3.2	Future Work	34
A	Real-time event simulation with frame-based cameras	37
B	eWand: A calibration framework for wide baseline camera systems	45
C	BiasBench: A reproducible benchmark for tuning the biases of event cameras	53
D	Detection of Fast-Moving Objects with Neuromorphic Hardware	69
E	Table tennis ball spin estimation with an event camera	79
F	An Event-Based Perception Pipeline for a Table Tennis Robot	91

Bibliography	101
---------------------	------------

Chapter 1

Introduction

Table tennis robots have gained an increased amount of interest in the research community. This is evident, given the number of published works in recent years (Gomez-Gonzalez *et al.*, 2019; Tebbe *et al.*, 2019; Ding *et al.*, 2022; D'Ambrosio *et al.*, 2023; D'Ambrosio *et al.*, 2024). The main reason is that table tennis robots are an ideal use case to push perception and control algorithms to their limits. The fast-paced sport does indeed present a challenge for the robotics system. Presently, the best table tennis robots are not yet able to compete with professional players, but can compete with amateurs and players with intermediate levels (D'Ambrosio *et al.*, 2024).

So far, all table tennis robots have used conventional, frame-based cameras for their perception pipeline. From decades of research, there exist mature algorithms for frame-based cameras. However, motion blur can render the task of ball detection difficult. One way to counteract this is by using a shorter exposure time and high frame-rate cameras. While having more frames per second allows algorithms to detect the flying table tennis ball more frequently, this also increases shot noise, power consumption, and the computational requirements. Therefore, the bottleneck is shifted from the sensor to the algorithm.

An alternative approach is to use event-based cameras. Event-based cameras (Gallego *et al.*, 2022) do not record all the pixels at a fixed frequency, but they report the change in illumination intensity for every pixel independently and asynchronously, visualized in Fig. 1.1. This results in an event stream with very low latency and a time resolution on the order of μs . Since only changes in intensity are registered, static parts of the scene

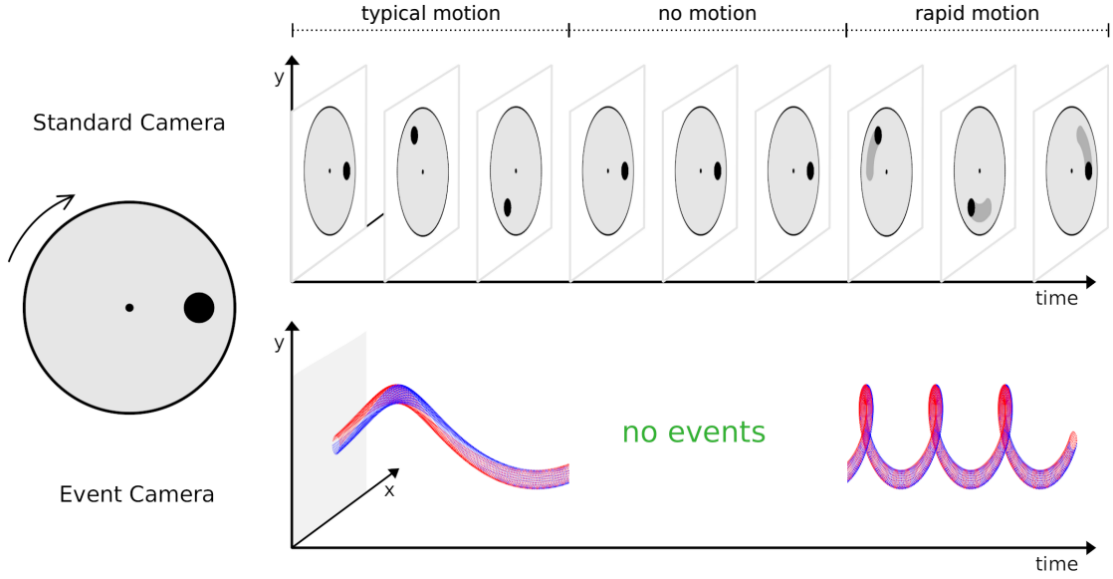


Figure 1.1: A visualization of the working principle of an event-based camera (Event Camera) compared to a frame-based camera (Standard Camera). While a frame-based camera records frames at a fixed interval, an event-based camera records a stream of events. Event-based cameras offer non-redundant data output, only informative pixels or no events. Red and blue dots represent *ON* and *OFF* events, respectively. Taken from Kim *et al.* (2016).

do not trigger events, which leads to a reduced level of data redundancy. Additionally, due to the logarithmic nature of the pixel’s photo-diode circuit, the event-based camera also has a high dynamic range, making them able to operate on a broad range of illuminations.

This makes the event-based camera a good fit for the perception pipeline of table tennis robots. In the following, we further motivate this PhD thesis and highlight the most important work this thesis is built on.

1.1 Motivation

With this thesis, *Event-Based Vision for Fast Robot Control*, we investigate how event-based computer vision can be used to control a table tennis robot, a system where fast processing times are crucial.

This thesis examines several previous systems and technologies. In the following, we describe the table tennis robot used in Section 1.1.1 and

introduce event-based computer vision in Section 1.1.2. In Section 1.1.3, we will give an overview of how we can combine event-based computer vision with the existing table tennis robot.

1.1.1 A table tennis robot system

The context of this thesis is a previously developed table tennis robot system (Tebbe *et al.*, 2019). This table tennis robot system has a high accuracy of vision detection and a fast robot reaction. It is based on an industrial KUKA Agilus R900 sixx robot with 6 DOF, shown in Fig. 1.2. Four frame-based cameras are used for ball position detection at 150 fps. To

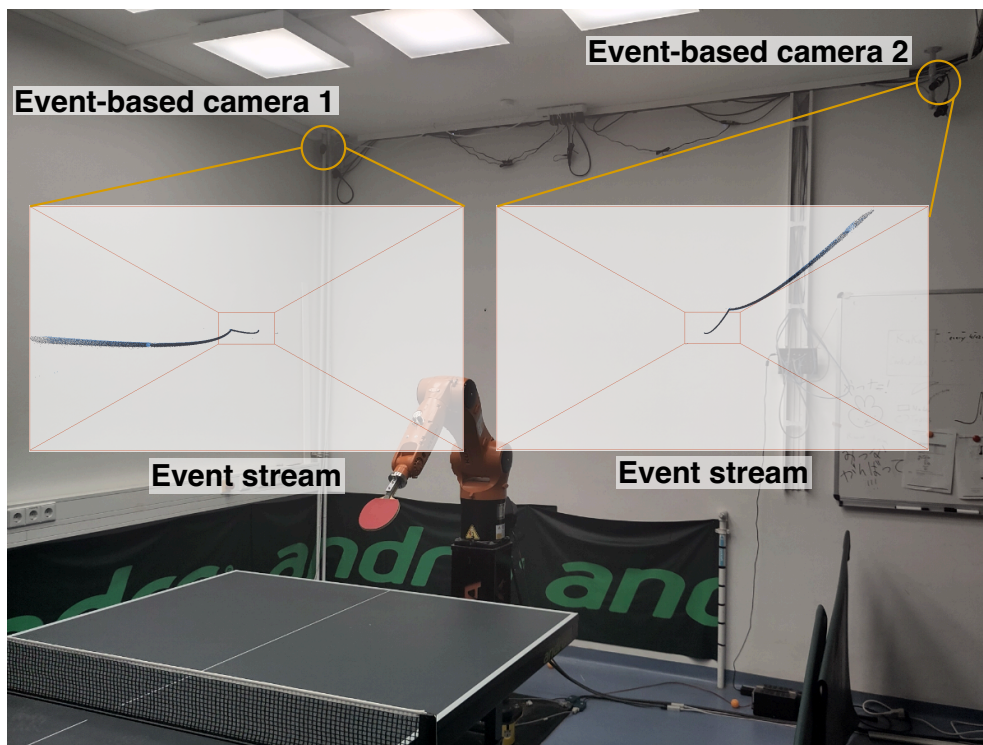


Figure 1.2: Our table tennis robot setup, consisting of a KUKA Agilus R900 sixx robot arm, four frame-based cameras (not visible in this picture), and two Prophesee EVKv4 event-based cameras. The two visualized event streams give a sense of the output of the event-based camera.

detect the flying ball with higher velocities in real-time, color and background thresholding are combined. This robot system is able to play rallies with a human counting up to 50 consecutive strokes and has a general hitting rate of 87%.

1.1.2 Event-based computer vision

Event-based cameras are bio-inspired sensors that differ from conventional frame cameras: Instead of capturing images at a fixed rate, they asynchronously measure per-pixel brightness changes and output a stream of events that encode the time, location, and sign of the brightness changes (Gallego *et al.*, 2022). Event-based cameras offer attractive properties compared to traditional cameras: high temporal resolution (on the order of μs), very high dynamic range (140 dB vs. 60 dB), low power consumption, and high pixel bandwidth (on the order of kHz) resulting in reduced motion blur, as shown in Fig. 1.1. Hence, event cameras have great potential for robotics and computer vision applications.

1.1.3 Event-based computer vision for fast robot control

The table tennis robot system mentioned earlier is an ideal platform for investigating the capabilities of event-based computer vision in a time-constrained environment.

Thus, in this thesis, we developed algorithms that take an event stream from statically mounted event-based cameras, visualized in Fig. 1.2, as input to get the same output as the frame-based perception system of the mentioned table tennis robot. Taking advantage of the beneficial properties of event-based cameras, we investigate whether event-based computer vision can achieve the same level of accuracy as the perception system based on conventional frame-based cameras. Furthermore, we investigated whether event-based computer vision provides benefits over conventional frame-based computer vision. Therefore, we analyzed how much the existing table tennis robot system can be improved using these novel kind of sensors and algorithms for perception.

The two main states that the perception system estimates are the position and the spin of the table tennis ball. Given enough 3D ball positions, the ball's trajectory can be predicted and used for robot control. The main focus of this PhD thesis is, therefore, event-based ball detection and spin estimation, given the system's temporal constraints. Using the advantages of event-based cameras, the potential benefits of such a perception system are the following:

Asynchronous data: Event-based cameras produce a stream of events as opposed to frame-based cameras, which sample the intensity of all the pixels at a given framerate. The asynchronous event stream allows us to constantly estimate the ball's position and spin, which should result in a smoother trajectory estimate. This can help the robot control part of the system since smoother trajectories could lead to faster convergence of the estimated hitting point.

Higher temporal resolution: Since timing is crucial in our use case, having position and spin estimates faster than with the frame-based perception system can give the robot control part more time. This could lead to a more accurate hitting point since the path planning and robot arm trajectory optimization have a more significant time budget.

1.2 Related Work

As explained in Section 1.1, this PhD thesis focuses on the perception part of a table tennis robot system. Therefore, we will not cover the literature on robot arm control.

The related work is split into multiple parts. We start with the related work on event-based camera simulation in Section 1.2.1. Although this is not the main topic of this thesis, our contribution of a real-time capable event simulator does fit within the theme of this PhD thesis. We will then briefly cover event-based and multi-modal camera calibration in Section 1.2.2. To work with our multi-modal wide baseline camera setup, we

had to develop our own camera calibration approach since the literature so far did not present any suitable method for our use case. For event-based cameras, tuning the settings, called *biases*, is crucial for any downstream application. In this thesis, we developed an algorithm to tune the biases of an event-based camera automatically. Therefore, we also briefly cover the related work on automatic bias tuning in Section 1.2.3. We will then cover the main research directions: ball detection, in Section 1.2.4, and spin estimation, in Section 1.2.5.

1.2.1 Event-based camera simulation

Event-based cameras are expensive sensors with a price tag of several thousand Euros. This is one of the motivations why event simulators are popular. A second, more technical reason is that simulators allow to generate a large amount of data relatively quickly from abundant video sequences. This allows the development of learning-based algorithms that rely on vast amounts of data, whether frame-based or event-based data.

One of the first event simulators was described in Kaiser *et al.* (2016). This event simulator subtracts consecutive frames and afterward applies a threshold to retain only pixels with a significant difference. The approach in Nehvi *et al.* (2021) is also based on subtracting frames. However, their contribution is the differentiability of the event simulator, which is needed for backpropagation. Approaches such as Mueggler *et al.* (2017); Li *et al.* (2018); Radomski *et al.* (2021) use a rendering engine to generate synthetic images from a 3D scene.

In Rebecq *et al.* (2018), the authors developed an event simulator tightly coupled with the rendering engine to adaptively sample the visual signal from the virtual camera sensor along a given camera trajectory. This allows the generation of simulated events asynchronously, whereas previous simulators only provide synchronous events. This approach also simulates some linear noise but neglects non-linear noise. In Gehrig *et al.* (2020), the authors extended their previous work (Rebecq *et al.*, 2018) by combining it with a frame interpolation method to generate events from videos. Compared to previous work, Hu *et al.* (2021) includes more realistic noise

models and, therefore, comes closer to the output of real event cameras. In Joubert *et al.* (2021), the authors go one step further and add several improvements and additions. The model notably improves the noise simulation by directly mapping noise distributions from real pixels. In another recent approach, the authors developed a GAN-style neural network to regress an event volume (an event representation) (Zhu *et al.*, 2019). Two consecutive frames and the corresponding events were given to train the network. On inference time, only two consecutive frames are needed. While the approach can approximate the true noise distributions of real events more accurately, it does not seem to run in real-time, and the output is an event representation and not the events themselves.

Most existing event simulators focus on the simulator’s realistic event output. This comes at the cost of a high computational load and, therefore, an increased runtime. For many applications, especially learning-based solutions, such event simulators are ideal. However, many robotics applications, especially closed-loop robotic systems, favor real-time capability over realistic events with accurate noise.

1.2.2 Event-based camera calibration

Since calibration of conventional cameras with printed calibration targets is a well-established topic with several toolboxes freely and commercially available (Rehder *et al.*, 2016; Zhang, 1999; Tangram, 2023), we focused on the calibration of event-based cameras, multi-modal calibration methods, and wand-based calibration methods.

There are two methodologies for event-based camera calibration: either frames are reconstructed from the event stream, and then traditional calibration methods can be used, or the events are directly used for the calibration. The first approach was introduced by Muglikar *et al.* (2021). In this work, E2VID (Rebecq *et al.*, 2019) is used to reconstruct frames from the event stream. With the reconstructed frames, the Kalibr calibration toolbox (Rehder *et al.*, 2016) is used for the calibration, similarly to conventional cameras. This method has the advantage that both event-based and frame-based cameras can be calibrated at the same time. The

frames of the frame-based camera(s) and the events can be recorded synchronously, and then, the reconstructed frames from the event-based camera(s), together with the conventional frames, can be fed into any camera calibration toolbox. In such a setup, it is important that the event-based and frame-based cameras are time-synchronized.

Early works on camera calibration for event-based cameras make use of blinking LED patterns (Mueggler and Gehrig, 2014; Dominguez-Morales *et al.*, 2019) or blinking screens (Mueggler *et al.*, 2014; VLOGroup, 2017; Prohse, 2022) to detect and extract the calibration patterns. Given the positions of the calibration targets, standard optimization-based methods can be used for the calibration (Muglikar *et al.*, 2021). Next to the need for a custom-built calibration target, these methods have the disadvantage that extrinsic calibration to frame-based cameras is not feasible (Muglikar *et al.*, 2021). Additionally, and even more importantly, our method allows for calibrating both sensor modalities together.

In a more recent work, Huang *et al.* (2021) introduced a dynamic event-based camera calibration algorithm. The approach directly calibrates, given events captured during relative motion between the camera and the calibration pattern. The method is propelled by a novel feature extraction mechanism for calibration patterns and leverages existing calibration tools before optimizing all parameters through a multi-segment continuous-time formulation. Although the calibration approach presented in Huang *et al.* (2021) does use a circular pattern, often used for the calibration of conventional cameras, it is not suited to calibrate event-based and frame-based cameras simultaneously.

While most of the camera calibration literature and available calibration toolboxes use printed calibration targets, there also exist methods that use a wand as a calibration target (Mitchelson and Hilton, 2003; Shin and Mun, 2012). Using a wand as a calibration target allows rapid, flexible, and accurate calibration of multiple camera systems and does not require the cameras to all face in the same direction or have overlapping fields of view (Mitchelson and Hilton, 2003). Similar to printed calibration targets, the position of the wand has to be detected in the frames in the first step. After the position of the markers on the wand is extracted, an optimization

algorithm estimates the intrinsics and extrinsics of the camera system by minimizing the reprojection error. While these methods were successfully applied to multi-camera systems with frame-based cameras, none of them could be used with event-based cameras directly.

1.2.3 Bias tuning for event cameras

As with any camera, the output quality depends on how well the camera's settings, called *biases* for event-based cameras, are configured. The biases of event-based cameras refer to a set of programmable current and voltage parameters that govern how each pixel in the sensor reacts to light changes. These biases critically shape the sensor's behavior in terms of sensitivity, bandwidth, noise characteristics, and maximum event rate (Graça *et al.*, 2023; Rui Graca *et al.*, 2023). Below is a general overview of the bias names and their functions, adapted from Prophesee's Metavision documentation¹.

- **bias_diff_on:** Sets the contrast threshold for *ON* events. Lower values make the pixel more sensitive to increases in brightness.
- **bias_diff_off:** Sets the contrast threshold for *OFF* events. Lower values increase sensitivity to decreases in brightness.
- **bias_refr:** Controls the duration the pixel is unresponsive after firing (prevents back-to-back events).
- **bias_fo:** Controls the low-pass filter. This changes how rapidly fluctuating light is filtered out. It determines the maximum rate of change of illumination that can be detected.
- **bias_hpf:** Controls the high-pass filter. This determines how slow illumination change is filtered out. It determines the minimum rate at which the light must change for the pixel to output an event.

While frame-based cameras have advanced automatic configuration algorithms, there are few such tools for tuning these *biases*.

¹<https://docs.prophesee.ai/4.6.2/hw/manuals/biases.html?highlight=bias>

Several papers meticulously document the behavior of the event-based camera. In Graça *et al.* (2023), the behavior of the DVS pixels for different bias parameters is described through extensive experimentation and measurements. In addition, the authors proposed a new way to interpret events as a combination of signal, noise, and leakage. An outline of the manual bias optimization process is given, and general indications are proposed for different scenarios. This, and several other papers (Gossard *et al.*, 2024b; Rui Graca *et al.*, 2023), have described the process of manually finding optimal biases. The most widely used metric in the literature for rating an event camera’s output is the Event Rate (ER) (Delbruck *et al.*, 2021; Nair *et al.*, 2024). However, the ER heavily depends on the observed scene and is therefore ambiguous when comparing different scenes.

So far, tuning the biases has had to be done manually. Other works have simply used the camera’s default settings (Amir *et al.*, 2017) or do not mention them (Gehrig *et al.*, 2021).

Comparatively, little has been done in the field of automatic bias tuning, given the plethora of research around event-based computer vision. Multiple papers (Graça *et al.*, 2023; Gossard *et al.*, 2024b; Nair *et al.*, 2024) dealing with the problem of finding optimal bias settings have resorted to measuring and rating the output for a range of different bias settings and then manually changing these biases.

In Delbruck *et al.* (2021), the authors presented a closed-loop control method, representing a first step towards automatic bias tuning, using a combination of three fixed-step feedback controllers. These feedback controllers take measurements of the ER and noise and regulate them using all five bias parameters. An optimal range is defined for the ER and the noise, and a simple algorithm changes the bias values if the two measurements leave their optimal ranges. The ER is regulated by increasing the thresholds if it is too high and decreasing them if it is too low. The ER is also limited by increasing the refractory period. The noise is controlled by adjusting the bandwidth. One limitation is that achieving the optimal bias setting requires multiple steps. Additionally, each threshold adjustment can trigger a burst of events, potentially leading to a short-term decline in

the quality of the event stream with every change.

Another approach, and the most advanced work on the topic of which we are aware, builds on the same idea but uses a two-part algorithm (Nair *et al.*, 2024). The algorithm controlling the biases uses two methods: fast adaptation, which changes the refractory period in order to limit the ER, and slow adaptation, which consists of changing the thresholds and adjusting the low-pass filter if fast adaptation is not enough to limit the ER. This split into two adaptations is necessary because only the refractory period does not cause visual artifacts when changed, which is problematic for the visual place recognition task presented in Nair *et al.* (2024). This means that the other three bias parameters should not be changed too often so as not to cause too much noise. For downstream applications such as visual(-inertial) odometry, SLAM, or object detection, this could lead to missing objects or losing track of the trajectory. While this approach is more advanced, it only considers the ER as a metric for the quality of the event stream.

1.2.4 Ball detection

Ball detection is a crucial part of a table tennis robot system. There are already several approaches in the literature (Tebbe *et al.*, 2019; Liu *et al.*, 2014; Ding *et al.*, 2022; Ji *et al.*, 2021; Gomez-Gonzalez *et al.*, 2019). However, none of them use event-based cameras.

Compared to learning-based methods, the literature about model-based object detection with event data is scarce. Although learning-based approaches have shown state-of-the-art detection accuracy (Perot *et al.*, 2020; Schaefer *et al.*, 2022; Liu *et al.*, 2024), they have inference times that are higher than what we need for our use case.

In Glover and Bartolozzi (2017), the authors presented a particle-filter-based tracking system. Their model assumes a static object, which requires the filter to be updated with a higher frequency than the object moves. The estimated state is the window size. Within the window size, events are accumulated, and the object is tracked. Compared to a previous approach with a Hough-filter (Glover and Bartolozzi, 2016), the particle

filter also manages to track the object with low velocities (because the window size is increased in these cases). The particles need a measurement function for the update step, which returns the probability of the tracked object. In this paper, they tracked a ball and, therefore, developed a heuristic model for a ball. Other heuristics would need to be developed to track other objects. The authors in Glover and Bartolozzi (2017) mentioned that the size of the ball tends to be underestimated, and a better measurement function might increase the accuracy.

In Forrai *et al.* (2023) and Falanga *et al.* (2020), events are clustered using DBSCAN to differentiate between the background and independently moving objects with a motion-compensated mean time stamp image. In our setup, the cameras are static, and therefore, motion compensation, which tends to be computationally heavy, is not needed.

1.2.5 Spin estimation

Spin plays a considerable role in table tennis, making a shot's trajectory harder to read and predict (Gossard *et al.*, 2023). There is very little literature on spin estimation for table tennis robots. Only a few works, e.g., Tebbe *et al.* (2020), explicitly estimate the ball's spin and take it into consideration when having the robot play.

The research on spin estimation conducted by our research group (Tebbe *et al.*, 2020; Gossard *et al.*, 2023) uses frame-based cameras running at 150 fps. The authors of Nakabayashi *et al.* (2024) used simulated event data and the Contrast Maximization (CMax) framework (Gallego *et al.*, 2019) to show that the spin of a ball can be estimated using event data. However, relying on simulated data raises the question about the sim-to-real gap. Another drawback of Nakabayashi *et al.* (2024) is the fact that it is an offline method, and therefore, not able to run in a real-time robotic setup.

1.3 Problem Statement

Similar to the related work, in Section 1.2, the problem statement is also split into multiple parts. We start with the problem statement of event-based camera simulation in Section 1.3.1. We then continue with the problem description of camera calibration in Section 1.3.2, followed by the research question on how to tune the biases of an event-based camera in Section 1.3.3. In Section 1.3.4 we define the ball detection. Afterwards, the problem of spin estimation is described in Section 1.3.5.

1.3.1 Event-based camera simulation

Event-based cameras are still expensive sensors, with a price tag of several thousand Euros for one camera. Simulators are an economical way to give the research community access to these new sensors. Additionally, simulators allow the generation of synthetic data that can be used for learning-based applications. The event simulators presented in the literature showed great performance for learning-based approaches but are too slow for closed-loop robotic applications.

Research Question 1: *Can an event-based camera simulator run in real-time?*

1.3.2 Camera calibration

As already mentioned in Section 1.2.2, camera calibration is a necessary first step to be able to do research with our multi-camera system.

Research Question 2: *How can a multi-modal (frame-based and event-based) camera system be calibrated in a practical way?*

1.3.3 Automatic bias tuning

Similar to camera calibration, optimal bias settings of an event-based camera are essential for a good performance of any downstream application.

Research Question 3: *How can the biases of an event-based camera be tuned in an automatic fashion?*

1.3.4 Ball detection

For the table tennis robot to hit the ball at the right time, position, and orientation, the trajectory of the ball needs to be estimated. The predicted trajectory contains the 3D positions $\{\tilde{x}_i, \tilde{y}_i, \tilde{z}_i\}$ for the time span when the ball is hit by a player until it touches the racket of the robot. For the trajectory prediction, the so far perceived 3D positions of the table tennis ball $\{x_i, y_i, z_i\}$ are needed (Tebbe *et al.*, 2019). Since this problem has already been solved with frame-based cameras, the focus is on using event-based computer vision approaches to detect the ball.

Research Question 4: *Can a table tennis ball detection system based on event-based computer vision improve the perception pipeline and, if so, how much?*

1.3.5 Spin estimation

As mentioned in Section 1.2.5, spin estimation can help to better estimate a ball's trajectory. In this thesis, we want to use event data to estimate the ball's spin.

Research Question 5: *Is it possible to estimate the spin of a table tennis ball given only event data?*

Chapter 2

Contributions

This chapter summarizes the key contributions of the papers that are attached in the appendix. It further highlights the connections between the individual results and refers to related work. In total, this research has been published in three peer-reviewed conference publications, two peer-reviewed conference workshop publications, and has led to one journal paper submission. Additionally, these works led to several open-source software repositories:

- Real-time event simulation with frame-based cameras
- eWand: A calibration framework for wide baseline frame-based and event-based camera systems
- Detection of Fast-Moving Objects with Neuromorphic Hardware
- BiasBench: A reproducible benchmark for tuning the biases of event cameras

2.1 Event-based camera simulation

Event-based cameras are still expensive sensors, with a price tag of several thousand Euros per camera. For other expensive sensors, such as LIDARs, simulators are an economical way to make the technology accessible to the research community. Simulators also have the advantage that data can be generated quickly and with less effort compared to real sensors. It is thus

no surprise that event simulators are frequently used for learning-based applications (Stoffregen *et al.*, 2020; Nakabayashi *et al.*, 2024).

The downside of this is that an accurate event stream is the main goal of existing event simulators, while runtime is not taken into account in the design decisions. However, for closed-loop robotic systems, simulators capable of running in real-time are crucial. Thus, we investigated whether it is possible to have an event simulator run in real-time and implemented our ideas as part of this PhD thesis.

2.1.1 Paper Appendix A: Real-time event simulation with frame-based cameras

Ziegler, A., Teigland, D., Tebbe, J., Gossard, T., and Zell, A. (2023b). Real-time event simulation with frame-based cameras. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE

To generate events between consecutive frames, most event simulators use frame interpolation, based on optical flow. The current state-of-the-art optical flow approaches are learning-based pipelines with high accuracy. Therefore, most event simulators use a learning-based optical flow pipeline, e.g., Super SloMo (Jiang *et al.*, 2018). The drawback is that this optical flow method has long run times, making it unusable for real-time applications. Indeed, most of the existing event simulators rely on offline frame interpolation before the actual event generation stage.

The contribution of our work is two-fold: We investigate optical flow methods running in real-time and use them to develop event simulation methods; we develop a novel event simulation method that leverages the sparsity of events in the interpolation to further improve the runtime.

By using the model-based optical flow methods Lucas Kanade (Lucas and Kanade, 1981), Farneback (Farneback, 2003), and DIS (Kroeger *et al.*, 2016), we reduced the runtime by several orders of magnitude while having comparable statistics for the event stream. Additionally, instead of interpolating between two consecutive frames, we introduced a difference frame interpolation approach, in which we interpolate between difference

frames. Meaning, given frames at the time stamps t_{i-2} , t_{i-1} , and t_i , we subtract the frame at t_{i-2} from the frame at t_{i-1} to get a first difference frame. We subtract the frame at t_{i-1} from the frame at t_i to get a second difference frame, shown in Fig. 2.1. These difference frames are motivated by events representing the change of intensity and not the absolute intensity. Difference frames tend to be sparser, especially for scenes with a low events-per-frame rate, and therefore, interpolating between difference frames can speed up the event simulation.

These contributions reduced the runtime by several orders of magnitude while having comparable statistics of the event stream. The results of our approaches for high events-per-frame data are presented in Fig. 2.2 and for low events-per-frame data in Fig. 2.3.

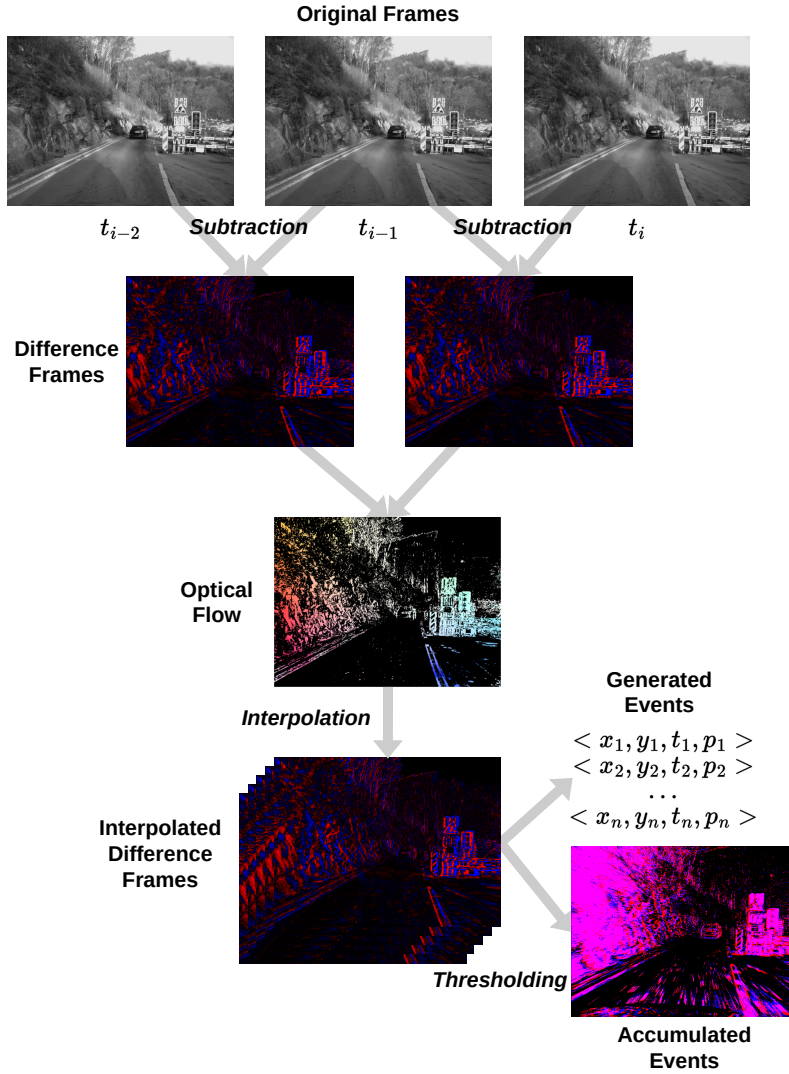


Figure 2.1: The process of the difference frame interpolation method. Given consecutive camera frames at t_{i-2} , t_{i-1} , and t_i , two difference frames are calculated in a first step. The optical flow between the two difference frames is calculated in the next step. With this optical flow, additional difference frames can be interpolated. In the last step, the interpolated difference frames are thresholded to determine all the pixels that should emit an event and the corresponding polarity of the event. The output is either the events in tuples of the form $\langle x, y, t, p \rangle$ with the location of the event (x, y) , the time stamp t and the polarity of the event p or a frame with the accumulated events. *Processing steps are written with italic letters* whereas (intermediate) outputs are written with standard letters.

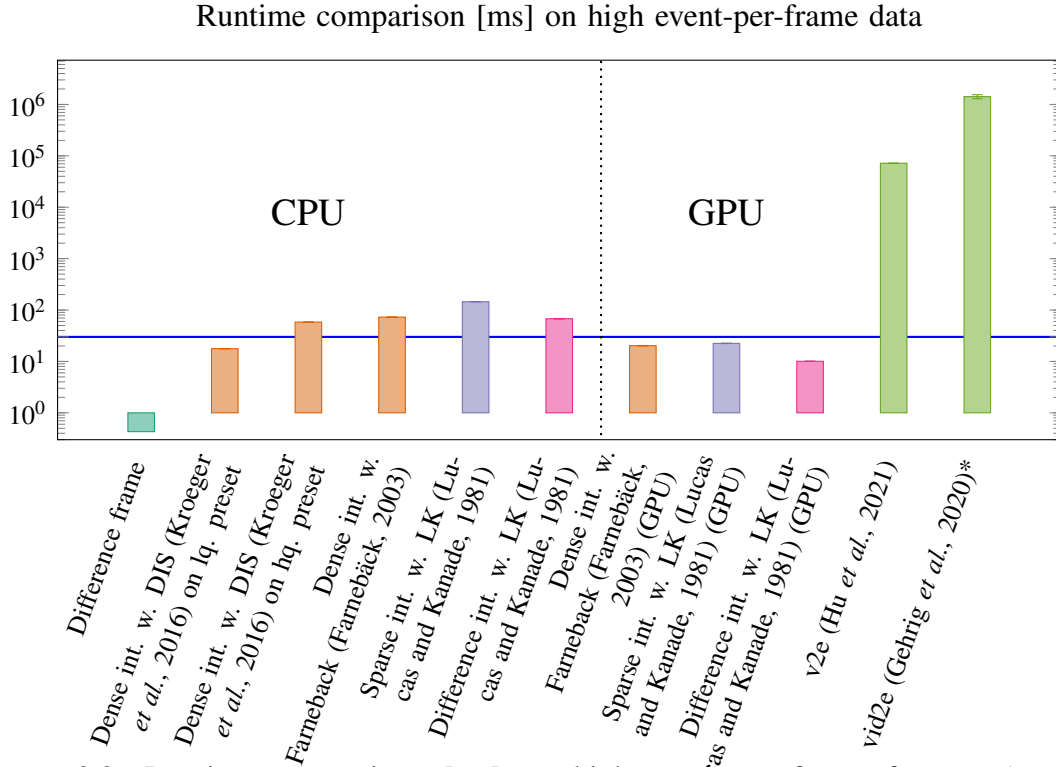


Figure 2.2: Runtime comparison [ms] on high events-per-frame footage (car in Zurich (Gehrig *et al.*, 2021)) with a resolution of 640×480 pixels. LK stands for Lucas Kanade (Lucas and Kanade, 1981). *For vid2e we only measured the time for the event generation but not for the upsampling due to the time-consuming upsampling. The bars visualize the mean value and the standard deviation (due to the log scale the standard deviation is barely visible). The blue line indicates a runtime of 30ms which we consider as real-time capable in this work.

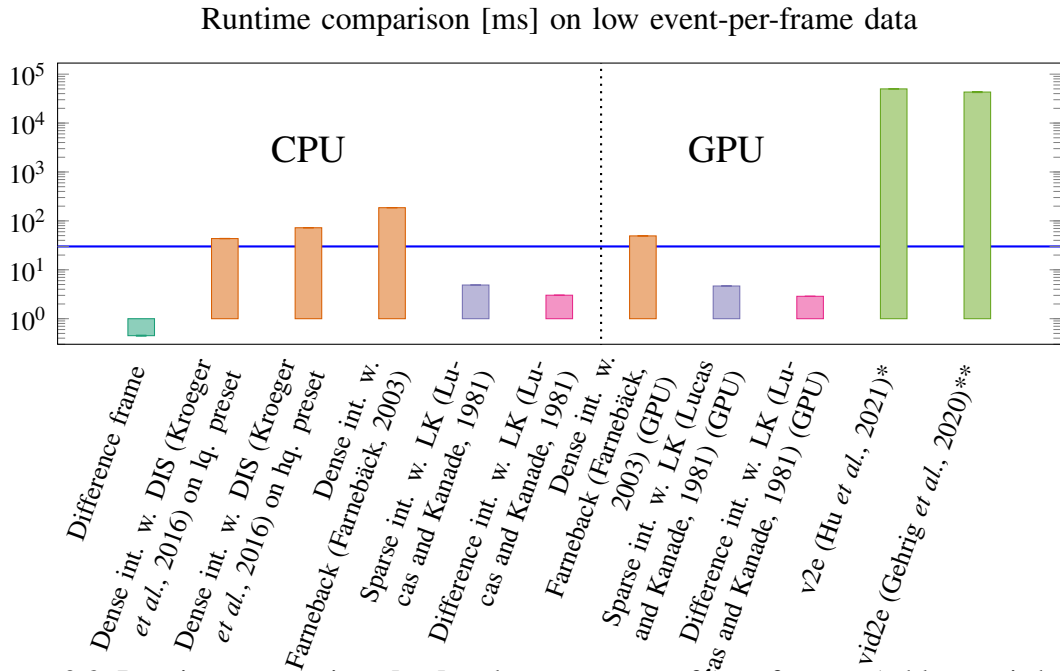


Figure 2.3: Runtime comparison [ms] on low events-per-frame footage (table-tennis ball) with a resolution of 1280×720 pixels. LK stands for Lucas Kanade (Lucas and Kanade, 1981). *Due to memory constraints on the GPU, we had to limit the output for v2e to 640×420 pixels. **For vid2e, we measured only the time for the event generation but not for the upsampling due to the time-consuming upsampling. The bars visualize the mean value and the standard deviation (due to the log scale the standard deviation is barely visible). The blue line indicates a runtime of 30ms, which we consider as real-time capable in this work.

2.2 Camera calibration

As mentioned in Section 1.3.4, we want to compare perception pipelines using event-based cameras with existing frame-based pipelines. Any multi-camera system needs to be calibrated intrinsically and extrinsically to be useful for tasks like triangulation. Additionally, it is beneficial to have only one coordinate system when comparing different perception systems. Therefore, camera calibration is necessary for conducting our research.

Our multi-camera system, containing frame-based and event-based cameras with wide baselines, is quite unique. Existing solutions are either not accurate enough or not practical. Therefore, we developed a novel calibration framework as part of this PhD thesis.

2.2.1 Paper Appendix B: eWand: A calibration framework for wide baseline frame-based and event-based camera systems

Gossard, T., Ziegler, A., Kolmar, L., Tebbe, J., and Zell, A. (2024a). ewand: A calibration framework for wide baseline frame-based and event-based camera systems. In *2024 International Conference on Robotics and Automation (ICRA)*. IEEE

For frame-based cameras, there is already a plethora of research about camera calibration, and mature calibration toolboxes are available. Since event-based cameras are still relatively new, there are few calibration solutions. Existing approaches can be divided into two categories: reconstructing frames from the event stream and then using existing calibration frameworks, or using blinking patterns that event-based cameras can detect and then optimizing the reprojection error. Both of these approaches have their drawbacks.

Generating frames from the event stream is usually done with E2VID (Rebecq *et al.*, 2019), a deep-learning-based pipeline that generates high-quality frames but at a high inference time. Thus, the frame generation is a time-consuming task. On the other hand, using blinking patterns does not require generating events but is unsuitable for frame-based cameras.

An additional challenge of our perception system is the wide baselines between cameras. For the intrinsic calibration, this does not cause a problem. However, extrinsic calibration makes matters more complicated. For an accurate extrinsic calibration, the calibration target should cover a substantial part of the camera’s Field of View (FoV), and this should be for as many cameras as possible. With baselines between 3 and 5 meters, this requires a calibration target with dimensions of more than a meter on each side. To prevent such a bigger calibration target from bending, which would decrease the accuracy of the calibration, the calibration target has to be reinforced. This results in a bulky and heavy calibration target, making the calibration procedure impractical.

Our work contributes threefold. We designed a wand with robust and accurate markers for frame- and event-based cameras, shown in Fig. 2.4. We developed detection methods for said markers. Extending a wand-



Figure 2.4: Our eWand, composed of three of our markers.

based extrinsic calibration to event-based cameras enables the extrinsic calibration of wide-baseline camera systems with both sensor modalities.

With our work, we achieved comparable reprojection errors, listed in Table 2.1, and 3D position errors, listed in Table 2.2, while significantly reducing the time needed for the calibration procedure.

Camera	BA eWand (our)	kalibr (circleboard)	kalibr (checkerboard)	BA circleboard	BA checkerboard
frame_0	0.325 ± 0.257	0.274 ± 0.060	0.390 ± 0.325	0.218 ± 0.219	0.225 ± 0.204
frame_1	0.320 ± 0.271	0.311 ± 0.070	0.398 ± 0.335	0.201 ± 0.178	0.201 ± 0.185
frame_2	0.401 ± 0.361	0.168 ± 0.072	0.320 ± 0.320	0.178 ± 0.178	0.213 ± 0.193
frame_3	0.396 ± 0.389	0.159 ± 0.062	0.301 ± 0.243	0.199 ± 0.188	0.180 ± 0.165
event_0	0.603 ± 0.448	0.269 ± 0.068	0.513 ± 0.396	0.398 ± 0.365	0.459 ± 0.363
event_1	0.550 ± 0.431	0.295 ± 0.070	0.515 ± 0.405	0.469 ± 0.447	0.402 ± 0.370

Table 2.1: Reprojection error (MAE) in pixels (mean and std.) for each calibration approach and camera, after the calibration. "BA" represents the bundle adjustment approach using OpenCV and Ceres.

Method	3D positional error [mm] (mean and std)
BA (checkerboard)	47.0 ± 35.8
BA (circleboard)	84.9 ± 35.8
kalibr (checkerboard)	8.4 ± 10.5
kalibr (circleboard)	8.6 ± 10.4
BA eWand (our)	9.3 ± 10.4

Table 2.2: The 3D position MAE [mm] (mean and std) of the different calibration methods. "BA" represents the bundle adjustment approach using OpenCV and Ceres.

2.3 Automatic bias tuning

For a camera to produce optimal output, its settings should be adjusted to the scene to be recorded. Event-based cameras are configured through five parameters, commonly referred to as *biases*, which govern event detection thresholds and signal filtering. These five *biases* are the *ON* and *OFF* thresholds for the event generation, the cut-off frequencies for a low- and high-pass filter, and the refractory period, which defines the dead time after a pixel has generated an event. While frame-based cameras have advanced automatic configuration algorithms, few such tools for tuning these *biases* exist.

Tuning these *biases* is necessary for downstream applications where a good signal-to-noise ratio is crucial. Manually tuning the *biases* is laborious and requires expertise. To alleviate this, we created a benchmark and a first baseline algorithm for tuning the *biases* of event cameras as part of this PhD thesis.

2.3.1 Paper Appendix C: BiasBench: A reproducible benchmark for tuning the biases of event cameras

Ziegler, A., Joseph, D., Gossard, T., Moldovan, E., and Zell, A. (2025a). Biasbench: A reproducible benchmark for tuning the biases of event cameras. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE

Bias tuning is a well-known challenge for researchers and practitioners working with event-based cameras and not relying on existing datasets. Although some work relies on the Event Rate (ER) to quantify the quality of an event stream (Delbruck *et al.*, 2021; Nair *et al.*, 2024), this does not transfer well between scenes. The ER can vary greatly depending on the dynamics in a scene and the amount of noise.

To develop algorithms for automatic bias tuning, a reproducible setup is needed where the effects of different settings can be evaluated. Recreating the same real-world recording setup in various research labs is difficult to achieve. As for other applications, event simulators could be considered as an option. However, the sim2real gap could cause problems, mainly because the biases depend on the electric circuit and the pixel design of the event camera, which are not accurately modeled in available event simulators. Therefore, we created a benchmarking dataset covering three different scenarios and a quality metric for each. Furthermore, we introduced an initial baseline algorithm for tuning the two most commonly used biases, the *ON* and *OFF* thresholds (controlling the sensitivity of the positive and the negative events, respectively), leveraging our dataset to enable online bias optimization.

2.4 Event-based ball detection

As stated in Section 1.3.4, one main contribution of this PhD thesis is an event-based perception pipeline for a table tennis robot. The first part of the perception pipeline of a table tennis robot is usually a ball detector. Detecting the ball in multiple calibrated cameras allows us to triangulate

the 3D position of the ball. With consecutive 3D ball positions, the trajectory of the ball can be predicted. In this PhD thesis, we developed two ball detection approaches. One classical, model-based approach and a learning-based approach which makes use of Spiking Neural Networks (SNNs) on Neuromorphic Hardware.

2.4.1 Paper Appendix F: An Event-Based Perception Pipeline for a Table Tennis Robot

Ziegler, A., Gossard, T., Glover, A., and Zell, A. (2025c). An event-based perception pipeline for a table tennis robot. *IEEE Robotics and Automation Letters*. to be submitted

Using event-based cameras for any computer vision or robotics tasks poses the question of which event representation to use. Many algorithms rely on some 2D representation since this is the common case for frame-based computer vision. With the events containing four dimensions (the spatial coordinates x and y , a time stamp t , and the polarity of the event p), the question is which information do we want to keep and which can we omit in a 2D representation. The next question is how to use the event data; there is a trade-off between processing events *event-by-event* and processing *batches-of-events*. While a single event does not contain enough information to be useful, processing it is usually fast. On the other hand, event batches contain enough information, but batching introduces a latency, and processing batches tends to be more time-consuming compared to single events.

An event representation that supports such an *event-by-event* update, without the need to recompute the whole representation, is the EROS event representation, used in Glover *et al.* (2024). EROS provides a coherent and bound spatial representation of the asynchronous events, partially maintaining information about their temporal order and attempts to capture the most up-to-date position of edges in the scene (Glover *et al.*, 2022).

Using EROS allows us to use the best of processing events *event-by-*

event and in *batches-of-events*. We do this by running two threads simultaneously. In the first thread, our perception pipeline processes the incoming events. It updates the EROS event surface *event-by-event*, shown in Fig. 2.5. In the second thread, the ball detection runs *as-fast-as-possible*, shown in Fig. 2.6.

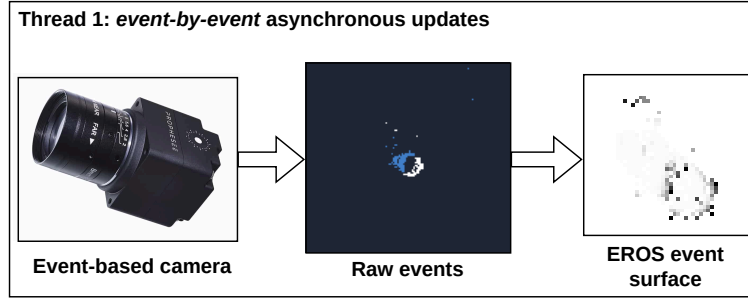


Figure 2.5: The first thread of our event-based perception pipeline. We process the incoming event and update the EROS event surface *event-by-event*.

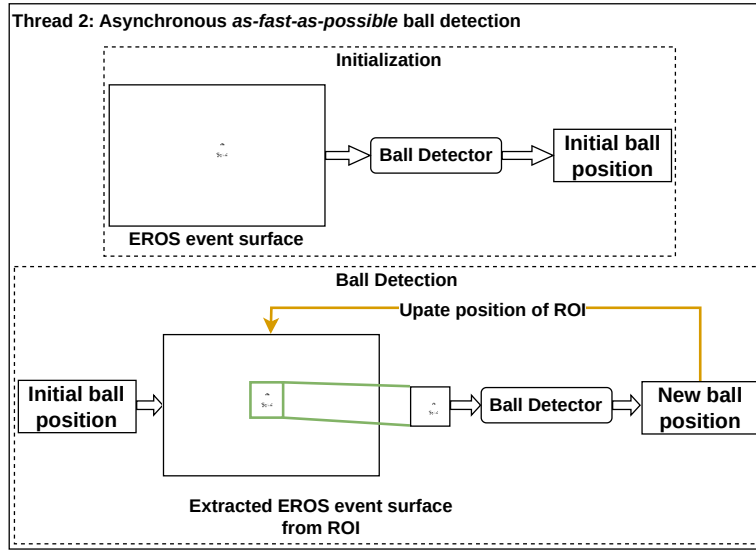


Figure 2.6: The second thread of our event-based perception pipeline. We initialize the Region of Interest (ROI) of the ball and, afterward, detect the ball within the ROI *as-fast-as-possible*.

With this approach, we achieve an update rate several orders of magnitude higher than frame-based pipelines while achieving a similar accu-

racy, reported in Table 2.3. We showed that the higher update rate results

Method	UR [updates/s]	RT [ms]	E [pixels]	IoU
FB baseline	149.0Hz*	0.8 (2.2)	1.32	$\triangle$
EB median filter bl	$(1.38 \pm 0.12) \times 10^4$	0.00008 (1.1)	2.90 ± 1.60	$\dagger$
EB particle filter bl	$(3.49 \pm 0.40) \times 10^3$	0.7 (1.0)	2.93 ± 1.13	0.55 ± 0.08
EB pipeline	$(4.14 \pm 0.10) \times 10^3$	0.1 (0.66)	1.34 ± 0.79	0.78 ± 0.11

Table 2.3: 'FB' stands for frame-based, 'EB' for event-based, 'bl' for baseline, 'UR' for update rate, 'RT' for average runtime, and 'E' for error. The result of the FB baseline is taken from Tebbe *et al.* (2019). The EB particle filter bl is a re-implementation of Glover and Bartolozzi (2017). The average run time in brackets is during the initialization. * Since the updates of the frame-based baseline are synchronous, in comparison to the asynchronous event-based pipelines, we report the result in Hz. $\triangle$ The authors in Tebbe *et al.* (2019) did not provide IoU results. $\dagger$ The event-based median filter baseline only provides the position, and therefore, it is not possible to calculate the IoU.

in lower estimated position, velocity, and spin uncertainties. This is beneficial for the path planning and robot control of a table tennis robot, which has to be fast to rally the ball back successfully.

2.4.2 Paper Appendix D: Detection of Fast-Moving Objects with Neuromorphic Hardware

Ziegler, A., Vetter, K., Gossard, T., Tebbe, J., Otte, S., and Zell, A. (2025b). Detection of fast-moving objects with neuromorphic hardware. In *2025 International Conference on Robotics and Automation (ICRA)*. IEEE

The *events* from an event-based camera match with the spike format of Spiking Neural Networks (SNNs). SNNs mimic the spiking behavior of biological neurons, offering a biologically inspired approach to Neural Network (NN) computation. Unlike traditional artificial neurons that produce real-valued outputs, spiking neurons receive input spikes and integrate them in the state of the neuron, called membrane potential. When the neuron's membrane potential reaches a defined threshold, the neuron emits a spike that propagates through the network, and resets its membrane potential.

With the availability of more computing power from GPUs, it became feasible to train and simulate SNNs for real-world tasks. However, simulating SNNs is inefficient on GPUs since the Neuromorphic Computing (NC) paradigm is fundamentally different. Therefore, running SNNs on GPUs, though possible, is not a reasonable option for real-world applications, especially not for real-time robotics applications. In contrast, neuromorphic hardware, specifically designed for efficient SNN processing, can leverage the sparsity and binary nature of SNN outputs, drastically reducing power consumption and latency compared to conventional NNs.

In this work, we explored the combination of an event-based camera and SNNs for object detection and their deployment on neuromorphic edge devices. We analyzed the potential benefits of this fusion and discussed the limitations of such deployment.

Three different SNN frameworks, namely sinabs, MetaTF, and Lava, and a corresponding neuromorphic edge device (DynapCNN, Akida, and Loihi2) were used, and their errors, time per forward pass, inference time, and power consumption were compared. The results are listed in Table 2.4 and Table 2.5. Moreover, we show that an SNN on a neuromorphic hard-

Framework	Sim	Hardware	Steps	Error [pixel]	One forward pass [ms]
BrainChip	✓		15 (1)	1.44 ± 1.41	1.75 ± 0.82
sinabs	✓		8 (1)	1.55 ± 1.89	1.59 ± 1.61
Lava	✓		20	1.57 ± 1.55	1764.89 ± 1374.36
BrainChip		Akida	15 (1)	1.44 ± 1.41	2.20 ± 0.35
sinabs		DynapCNN	8	1.59 ± 1.83	46.04 ± 21.38
Lava		Loihi2	20	1.57 ± 1.55	1458.57 ± 230.500
TensorFlow		RTX 2080 Ti	-	3.55 ± 1.49	3.49 ± 0.62

Table 2.4: Ball center error and time per forward pass of the different SNN frameworks and on their corresponding neuromorphic edge device. Steps indicate how many time steps were used for one inference pass. The results are reported with mean and standard deviation over multiple runs.

ware is able to run in a challenging table tennis robot setup in real-time. Despite the promise of asynchronous processing with SNNs, the current edge devices face limitations, primarily attributed to hardware integration.

Device	One forward pass [ms]	Inference time [ms]	Power consumption [mW]
Akida	2.20 ± 0.35	0.89 ± 0.28	~ 4.5
DynapCNN	$46.04 \pm 1.38^\dagger$	0.82 ± 0.24	$\sim 5^*$
Loihi2	1458.57 ± 230.50	0.62 ± 0.01	$\sim 100^*$
RTX 2080 Ti	3.49 ± 0.62	1.77 ± 0.28	$\sim 250000^*$

Table 2.5: Total time per forward pass and inference time of the different neuromorphic edge devices with mean and standard deviation over 10 runs, and the power consumption.

† Processing one input on the DynapCNN takes an average of 1.765ms, but with delays from the USB connection, the time for a forward pass is significantly longer. * Taken from specs.

2.5 Event-based spin estimation

While not every table tennis robot is using spin information explicitly (Ji *et al.*, 2021; D'Ambrosio *et al.*, 2023), previous research has shown that it is beneficial to model and perceive the spin of the ball (Tebbe *et al.*, 2019; Gossard *et al.*, 2023). Thus, spin estimation is often seen as the second major part of the perception pipeline of a table tennis robot. As with event-based ball detection, we also investigate event-based spin estimation as part of this PhD thesis.

2.5.1 Paper Appendix E: Table tennis ball spin estimation with an event camera

Gossard, T., Krismer, J., Ziegler, A., Tebbe, J., and Zell, A. (2024b). Table tennis ball spin estimation with an event camera. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, page 3347–3356. IEEE

Spin plays a pivotal role in ball-based sports. The fact that spin cannot be observed directly makes its estimation challenging. Among the different spin estimation approaches, logo-based spin estimation achieves the highest accuracy (Gossard *et al.*, 2023). Due to motion blur, the combination of high velocity and spin in table tennis renders conventional frame-based cameras inadequate for quickly and accurately observing the ball’s logo to

estimate the spin. Event-based cameras do not suffer as much from motion blur, thanks to their high temporal resolution. Moreover, the sparse nature of the event stream reduces data redundancy and the required bandwidth. In this work, we present the first table tennis spin estimation method using an event-based camera capable of running in real-time. We use ordinal time surfaces to track the ball and then isolate the events generated by the logo on the ball. Optical flow is then estimated from the extracted events to infer the ball's spin.

As can be seen in Fig. 2.7, for higher spin settings, we were able to compensate for the drawbacks of frame-based cameras with an event-based camera and achieve more reliable spin estimation. For lower spin settings, the frame-based approach performs better since only few events get triggered by the rotating ball logo. In table tennis, high spin values make the game challenging. Therefore, reliable spin estimation for higher spin values is beneficial. We achieved a spin magnitude mean error of 10.7 ± 17.3

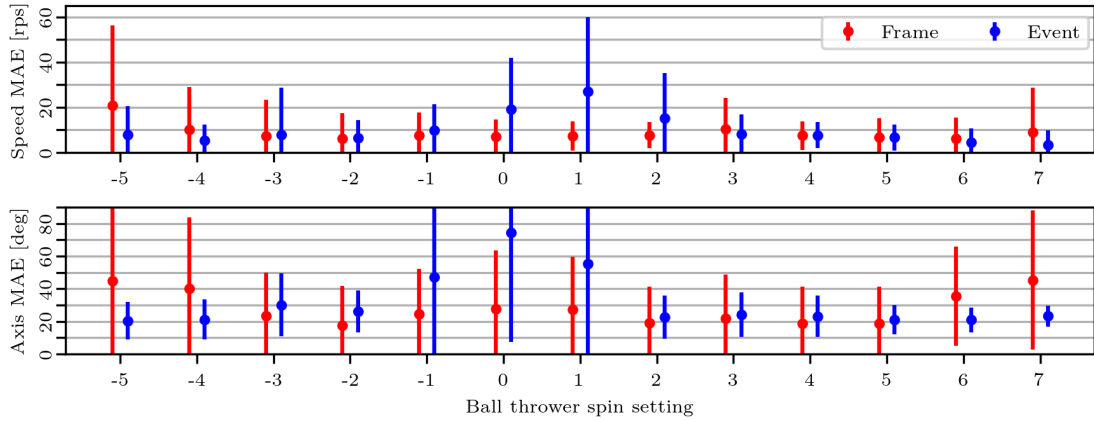


Figure 2.7: (top) Spin magnitude error with the ball thrower for different ball spins. (bottom) Spin axis estimation error with the ball thrower.

rps and a spin axis mean error of 32.9 ± 38.2 deg in real time for a flying ball.

2.6 Unrelated contributions

In this section, we briefly describe additional publications I contributed to during my PhD, even if they did not directly contribute to this thesis.

Gossard, T., Tebbe, J., Ziegler, A., and Zell, A. (2023). Spindoe: A ball spin estimation method for table tennis robot. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE

Gossard, T., Ziegler, A., , and Zell, A. (2025b). Tt3d: Table tennis 3d reconstruction. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE

Gossard, T., Schmalzl, J., Ziegler, A., and Zell, A. (2025a). Spin detection using racket bounce sounds in table tennis. In *2025 IEEE International Workshop on Sport, Technology and Research (STAR)*. IEEE

Thomas Gossard’s time with the Cognitive Systems Group overlapped with mine to a large extent. Both of us were funded by the same project, and, therefore, we also worked together closely. Although each of us had our own topics, we had many brainstorming sessions, shared ideas and maintained the hardware infrastructure in the table tennis lab together. Thus, I list Thomas Gossard’s main contributions during his PhD here as unrelated contributions.

Ziegler, A., Gossard, T., Vetter, K., Tebbe, J., and Zell, A. (2023a). A multi-modal table tennis robot system. In *RoboLetics: Workshop on Robot Learning in Athletics @CoRL 2023*

This contribution was a workshop paper in which we provided a snapshot of our recent work and results.

Chapter 3

Conclusion

Throughout this PhD thesis, we contributed to two fields and their intersection: event-based computer vision and table tennis robotics. In both, we achieved good results and new insights, but also realized that much work still needs to be done.

3.1 Conclusion

This PhD thesis started with a table tennis robot and one event-based camera. While we initially focused on event-based object detection—the first task of the perception pipeline of a table tennis robot—we quickly realized that some groundwork needed to be done first. This led to two contributions: one on camera calibration and another on automatic bias tuning for event-based cameras.

In parallel, a side project on simulating events from a frame-based camera evolved into a valuable contribution in its own right. Although it began independently, the resulting real-time event simulator aligned well with the overarching theme of enabling event-based vision for robotics and thus became part of this PhD thesis.

The main contributions of this thesis demonstrate the advantages of an event-based perception pipeline for a table tennis robot. We developed two ball detection approaches, one model-based and one learning-based. The model-based approach achieved an update rate several orders of magnitude higher than comparable frame-based systems, which reduces trajectory and spin uncertainties and benefits robot control.

The learning-based approach further demonstrated the suitability of event data for Spiking Neural Networks (SNNs) and showed that such networks can operate in real time on neuromorphic hardware within our closed-loop robotic system.

A second major contribution is an event-based spin estimation pipeline. We proposed a model-based algorithm that exploits the high temporal resolution of event data and outperforms frame-based solutions, particularly for fast-spinning balls.

3.2 Future Work

PhD research is necessarily limited to a given period of time and I had to realize the limitations of what can be achieved within this time. There are still some remaining open questions, as well as new ones that came up during this PhD thesis.

The algorithms developed in this thesis were either model-based or run on special, neuromorphic hardware. The motivation was that although GPUs enable heavy parallelization and learning-based methods, transferring data, event data in our case, from the CPU to the GPU usually takes around 1 ms. In contrast, model-based algorithms can detect the ball in under 1 ms. Newer GPUs and embedded GPU solutions might lead to improvements on this front and allow for learning-based algorithms with similar runtimes as model-based algorithms.

With working frame-based and event-based perception pipelines for a table tennis robot, another area for future research is combining them to get the best from both technologies. A first idea would be to use Extended Kalman Filters to fuse the results of both pipelines in a Bayesian fashion. Another idea is to combine the two modalities as input for a learning-based solution. There is literature in this direction; however, the majority put their focus on accuracy and not on speed.

Despite these remaining challenges, the ongoing progress in both hardware and algorithmic design offers exciting opportunities for future advancements, pushing the boundaries of real-time, robust perception sys-

tems for robotics by using event-based computer vision.

Appendix A

Real-time event simulation with frame-based cameras

Authors

Andreas Ziegler, Daniel Teigland, Jonas Tebbe, Thomas Gossard and
Andreas Zell

Published in

2023 IEEE International Conference on Robotics and Automation
(ICRA 2023)

Time of publication

June 2023

DOI

10.1109/ICRA48891.2023.10160654

Included in this dissertation with permission from IEEE.
©2023 IEEE.

Real-time event simulation with frame-based cameras

Andreas Ziegler, Daniel Teigland, Jonas Tebbe, Thomas Gossard and Andreas Zell

Abstract—Event cameras are becoming increasingly popular in robotics and computer vision due to their beneficial properties, e.g., high temporal resolution, high bandwidth, almost no motion blur, and low power consumption. However, these cameras remain expensive and scarce in the market, making them inaccessible to the majority. Using event simulators minimizes the need for real event cameras to develop novel algorithms. However, due to the computational complexity of the simulation, the event streams of existing simulators cannot be generated in real-time but rather have to be pre-calculated from existing video sequences or pre-rendered and then simulated from a virtual 3D scene. Although these offline generated event streams can be used as training data for learning tasks, all response time dependent applications cannot benefit from these simulators yet, as they still require an actual event camera. This work proposes simulation methods that improve the performance of event simulation by two orders of magnitude (making them real-time capable) while remaining competitive in the quality assessment.

SUPPLEMENTARY MATERIAL

The project's code and additional resources are available at https://cogsys-tuebingen.github.io/realtime_event_simulator/

I. INTRODUCTION

In the field of robotics and computer vision, event cameras have been rapidly gaining in popularity. Event cameras work fundamentally differently from traditional frame-based cameras. Instead of capturing an entire frame of pixels at once, an event camera registers log-scale changes in brightness on a pixel level. If the change is large enough, the camera emits an asynchronous event. The advantages of event cameras are that they can register events just microseconds apart, resulting in lower latency and higher temporal resolution (μs instead of ms) compared to frame-based cameras. These cameras also come with other great potential benefits, such as a high dynamic range and low power consumption [1]. This technology allows for super low latency applications that, when implemented correctly, can leverage all this data to outperform the best traditional computer vision algorithms in various robotics and computer vision tasks. Some useful applications include self-driving cars [2], general purpose object tracking [3], image segmentation [4], [5], high frame rate video reconstruction [6], accurate depth perception [7] and advanced frame interpolation with optical flow reconstruction [8] and robotics applications like control tasks [9], [10],

The authors are with the Cognitive Systems Group, Dept. Informatics, University of Tuebingen. Corresponding author andreas.ziegler@uni-tuebingen.de. This research was funded by Sony AI.

and odometry [11]. For some applications, e.g., self-driving cars [2], it is essential that this data is processed in real-time.

Although the first event camera was released more than a decade ago, event cameras remain expensive ($\sim 6000\text{€}$) and scarce in the market, making them inaccessible to many researchers, companies, and hobbyists alike. This work proposes using frame-based cameras to simulate this event data. Traditional frame-based cameras remain cheap and readily available, making this approach an interesting alternative.

Over the last few years, several methods have been published that attempt to simulate event data. While these recent publications produce great results of high quality in the event simulation, the computation required for just one frame can take up to multiple seconds, eliminating the possibility of real-time use. To truly use a frame-based camera as a replacement for an event camera in robotics applications, these simulators would have to run at much higher speeds.

This work specifically attempts to address this problem by presenting methods that aim to drastically improve the runtime of these simulations while remaining competitive in quality, to be of use for real-time robotics applications.

Contributions of this work are as follows:

- Optical flow based event simulation methods running in real-time
- A novel event simulation method which leverages the sparsity of events in the interpolation to further improve the runtime
- Qualitative and quantitative results comparing our event simulation methods with existing ones and real event cameras, and a guideline, when to use which simulator

The rest of the paper is organized as follows: Section II presents the related work. In Section III we present our methods. In Section IV we present qualitative and quantitative comparisons. Conclusions are drawn in Section V.

II. RELATED WORK

One of the first event simulators was described in [12]. This event simulator subtracts consecutive frames and afterward applies a threshold to retain only pixels that have a significant difference. The approach in [13] is also based on subtracting frames, however, their contribution is the differentiability of the event simulator, which is needed for backpropagation. While our approach subtracts consecutive frames, we use optical flow to interpolate the given input frames, leading to a temporal resolution closer to one of the real event cameras. Approaches such as [14], [15], and [16] use a rendering engine to capture images from a 3D scene.

In [17], the authors developed an event simulator tightly coupled with the rendering engine to adaptively sample

the visual signal from the virtual camera sensor along a given camera trajectory. This allows generating simulated events asynchronously, whereas previous simulators only provide synchronous events. This approach also simulates some linear noise but neglects non-linear noise. In [18], the authors extended their previous work [17] by combining it with a frame interpolation method to generate events from videos. Similar to [18], [19] converts regular video sequences to events. Compared to previous work, [19] includes more realistic noise models and therefore comes closer to the output of real event cameras. In [20], the authors go one step further and add several improvements and additions. The model notably improves the noise simulation by directly mapping noise distributions from real pixels. In another recent approach, the authors developed a GAN-style neural network to regress an event volume (an event representation) [21]. To train the network, two consecutive frames and the corresponding events were given. On inference time, only two consecutive frames are needed. While the approach can approximate the true noise distributions of real events more accurately, it does not seem to run in real-time, and the output is an event representation and not the events themselves.

Most of the existing event simulators focus on the realistic event output of the simulator. This comes at the cost of a high computational load, and therefore an increased runtime. For many applications, especially learning-based solutions, such event simulators are ideal. However, many robotics applications favor real-time capability over realistic events with accurate noise. Our event simulators have a less realistic noise model, but do run in real-time while remaining competitive in quality. To the best of our knowledge, this work presents the first method to simulate events given the output of a frame-based camera in real-time.

III. METHOD

We present three different methods to simulate events. All of them are based on the subtraction of consecutive frames, as done in [12]. We call the result of the subtraction the *difference frame*. In contrast to event cameras, we do not use the log difference but the linear difference. The quality improvement of log differences is small and comes with a high computational load (delay in ms depending on the interpolated frames). Further, this work does not address the noise and artifacts present in data from a real event camera.

The temporal resolution of frame-based cameras is much lower than that of event cameras [1]. To counteract this, we use interpolation to artificially increase the frame rate. Interpolation is a common step in event simulation and is also used in [18] and [19]. For interpolation, we use optical flow algorithms and a simple but fast linear interpolation approach. We used the fastest optical flow algorithms evaluated in [22].

Next follows the detailed description of the three methods. In Section III-A, we describe a method based on dense frame interpolation. Section III-B introduces a method that uses sparse frame interpolation, and the technique described in Section III-C interpolates only the *difference frames*.

A. DENSE FRAME INTERPOLATION

This method calculates the dense optical flow between two consecutive camera frames in the first step. This dense optical flow is then used to interpolate between the two consecutive camera frames. Given these interpolated frames, consecutive frames are subtracted to get *difference frames*. In the last step, the *difference frames* are thresholded (with C_{pos} and C_{neg}) to determine all the pixels that should emit an event, and the corresponding polarity of the event according to

$$e_{\text{pos}}(x, y) = \begin{cases} \text{pos. event,} & \text{if difference_frame}(x, y) > C_{\text{pos}} \\ \text{no event,} & \text{otherwise} \end{cases}$$

$$e_{\text{neg}}(x, y) = \begin{cases} \text{neg. event,} & \text{if difference_frame}(x, y) < C_{\text{neg}} \\ \text{no event,} & \text{otherwise.} \end{cases}$$

The process of this approach is illustrated in Fig. 1.

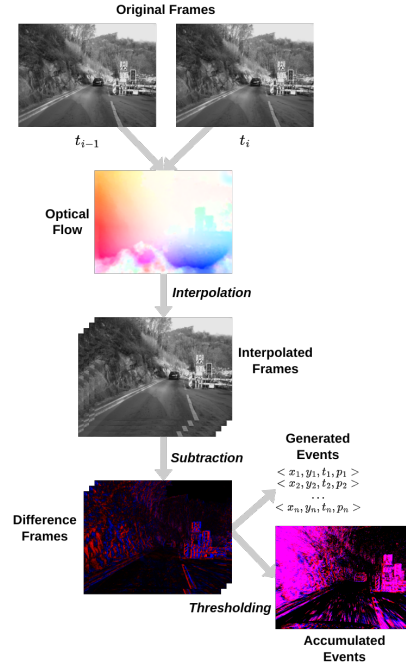


Fig. 1. The process of the dense frame interpolation method, given the consecutive camera frames at t_{i-1} and t_i as input. The output is either the events in tuples of the form $\langle x, y, t, p \rangle$ with the location of the event (x, y) , the time stamp t and the polarity of the event p or a frame with the accumulated events. *Processing steps* are written with italic letters whereas (intermediate) outputs are written with normal letters. For more details we refer to Section III-A.

The output is either the events in tuples of the form $\langle x, y, t, p \rangle$ with the location of the event (x, y) , the time stamp t and the polarity of the event p or a frame with the accumulated events.

In order to use this algorithm in real-time applications, the choice of the dense optical flow algorithm is very important, as its speed is the main bottleneck for the performance of this method.

B. SPARSE FRAME INTERPOLATION

This method is similar to the dense frame interpolation approach described in Section III-A with the difference that sparse optical flow is used to interpolate between two consecutive camera frames.

The motivation to use sparse optical flow is to increase the performance by reducing the number of calculations for the frame interpolation step. Dense optical flow algorithms consider the entire frame while only a fraction of the pixels in a frame is actually changing a lot, most of the time. Real event cameras use a threshold to detect when an event at a particular pixel has occurred. Analogously to an event camera, all pixels that have not changed within a given threshold from one frame to the next can be disregarded for the optical flow estimation. This way, the computation can be sped up as long as the number of pixels that have changed is substantially lower than the total number of pixels in the frame. The selected pixels can be estimated with any sparse optical flow algorithm.

It is important to note that sparse optical flow estimations are usually slower than dense algorithms per pixel, so this algorithm benefits from static backgrounds containing small moving subjects.

C. DIFFERENCE FRAME INTERPOLATION

Interpolating regular frames is a complex problem with many unforeseen edge cases such as, e.g., occlusions, and is computationally expensive. Traditional interpolation algorithms all rely on the basic assumption that the area of pixels that make up an object will not change drastically in color, brightness, or size from one frame to another. This assumption breaks down in scenes where occlusions occur. Traditional optical flow algorithms cannot find the original pixels, and artifacts appear in the optical flow estimation.

To reduce the possibility of such edge cases and to further decrease the computational load of the interpolation step, this approach only interpolates between consecutive *difference frames* rather than between consecutive camera frames. This method reduces complexity by only interpolating between the events themselves since we assume that the *difference frames* represent the events.

Given three consecutive camera frames, two *difference frames* are calculated in the first step. The optical flow between the two *difference frames* is calculated in the next step. With this optical flow, additional *difference frames* can be interpolated between the two *difference frames*, from the first step. The process of this approach is illustrated in Fig. 2.

This method makes some strict assumptions about the original frame sequence. The objects' movement that generates events must be of constant linear velocity. Furthermore, the perspective of these objects in the video cannot change over time. Given these assumptions, this method is specially targeted for tasks like, e.g., object tracking.

IV. EVALUATION

We evaluate our proposed simulation methods in three different ways. We start in Section IV-A with a comparison

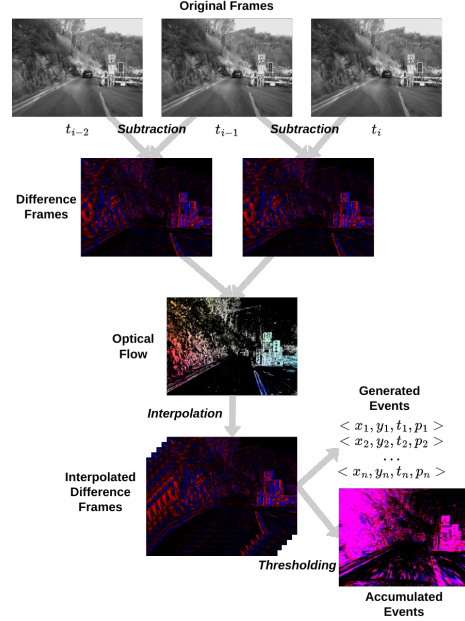


Fig. 2. The process of the difference frame interpolation method, given the consecutive camera frames at t_{i-2} , t_{i-1} and t_i as input. The output is either the events in tuples of the form $\langle x, y, t, p \rangle$ with the location of the event (x, y) , the time stamp t and the polarity of the event p or a frame with the accumulated events. *Processing steps are written with italic letters* whereas (intermediate) outputs are written with standard letters. For more details we refer to Section III-C.

of the runtimes of the different event simulation methods. In Section IV-B, we qualitatively compare the output of our methods to real events and the output of another event simulator. Next to the qualitative comparison, we compare some statistics of the simulated, the real events, and two other event simulators in Section IV-C.

A. RUNTIMES

We compare the runtimes of our proposed event simulation methods in two different scenarios. The first scenario is a dynamic scene with a high events-per-frame ratio. This footage is taken from [23]. The dataset contains recorded events and camera frames at 20 fps from a car driving around. We took a frame sequence with 537 frames from the *inter-laken_00_c* sequence. The frames with an original resolution of 1440×1080 pixels were resized to the resolution of the event camera (640×480 pixels) before passing it to the event simulators. For this dataset, we used 20 interpolated (difference) frames. The results of this experiment are shown in Fig. 3.

We use a less dynamic scene with a low events-per-frame ratio in the second scenario. Specifically, we recorded a flying table tennis ball in front of a static background with a FLIR CHAMELEON 3 camera at 150 fps and a Prophesee Gen4 event camera [24]. This frame sequence contains 158 frames. We also resized the input frames for this dataset from 1280×1024 pixels (the camera's resolution)

to 1280×720 pixels (the resolution of the event camera), before passing it to the event simulators. For this dataset, we used 10 interpolated (difference) frames. The results of this experiment are shown in Fig. 4.

Next to the runtimes of our three proposed methods, we also list the runtimes for calculating the *difference frame* as a reference. We also compare our event simulation methods to vid2e [18] and v2e [19], two state-of-the-art event simulators.

We used different optical flow algorithms for all three event simulation methods since the simulation methods heavily depend on the used optical flow implementation.

We chose C_{pos} , C_{neg} , and the number of interpolated frames so that the generated events resemble the ground truth accumulated event frame. These values are listed in Table I.

TABLE I
 C_{POS} , C_{NEG} VALUES AND THE NUMBER OF INTERPOLATED FRAMES

Method	C_{pos}	C_{neg}	# int. frames
Dense interpolation	2	-2	10
Sparse interpolation	10	-10	10
Difference interpolation	20	-20	10
v2e [19]	0.3	-0.3	?
vid2e [18]	0.2	-0.2	?

All the experiments were run on a desktop with an Intel i7 9700 3.0 GHz CPU, 32GB RAM, and an NVIDIA GeForce RTX 2080 Ti. For each algorithm, the video sequence was loaded into RAM before the time was measured. Then the entire video is passed to each algorithm frame by frame. After the last frame, the time measurement was stopped. We recorded the runtimes of 10 runs. From these, the mean runtime and the standard deviation were calculated.

Discussion As can be seen in the results shown in Fig. 3 and Fig. 4, the runtimes differ substantially depending on the event simulation method and their configuration.

Starting with the dynamic dataset, the dense interpolation method with DIS [22] on its lowest quality setting is the fastest configuration for the dense approach, closely followed by the one with Farneback [25] (on the GPU). DIS takes more than three times as long with the highest quality setting. The sparse interpolation method on the GPU is a bit slower than the dense one. The sparse interpolation method on the CPU performs the worst of all the techniques with different settings. Since there is a lot of movement in this driving car dataset, we expect the sparse interpolation method to be worse. As mentioned in Section III-B, sparse optical flow benefits only if the number of pixels that have changed is substantially lower than the total number of pixels. In this experiment, this is not the case. The GPU and CPU runtimes of the difference interpolation method are better compared to the corresponding runtimes of the sparse interpolation method, with the GPU version outperforming the CPU version.

On the less dynamic dataset, the runtimes for the dense interpolation method are all longer because the resolution of the table tennis video is larger (1280×720 pixels compared to 640×480 pixels). For the dense interpolation method, the relative comparisons remain the same. As can be seen

RUNTIME COMPARISON [MS] ON
HIGH EVENTS-PER-FRAME DATA

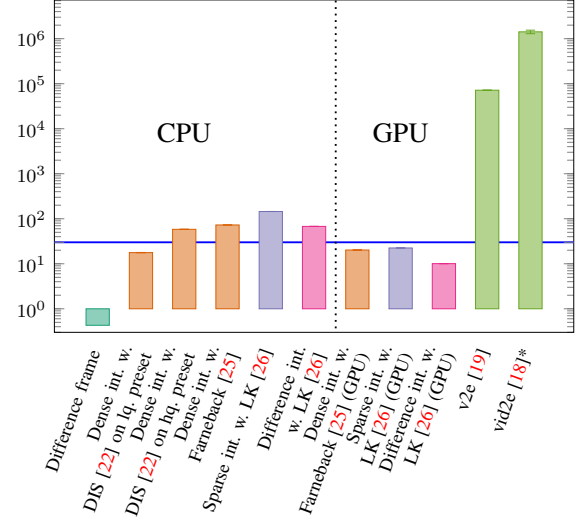


Fig. 3. Runtime comparison [ms] on high events-per-frame footage (car in Zurich [23]) with a resolution of 640×480 pixels. LK stands for Lucas Kanade [26]. *For vid2e we only measured the time for the event generation but not for the upsampling due to the time-consuming upsampling. The bars visualize the mean value and the standard deviation. The blue line indicates a runtime of 30ms which we consider as real-time capable in this work.

RUNTIME COMPARISON [MS] ON
LOW EVENTS-PER-FRAME DATA

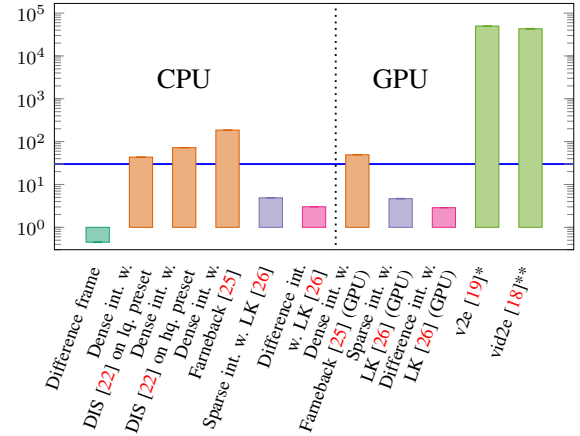


Fig. 4. Runtime comparison [ms] on low events-per-frame footage (table tennis ball) with a resolution of 1280×720 pixels. LK stands for Lucas Kanade [26]. *Due to memory constraints on the GPU, we had to limit the output for v2e to 640×420 pixels. **For vid2e we only measured the time for the event generation but not for the upsampling due to the time-consuming upsampling. The bars visualize the mean value and the standard deviation. The blue line indicates a runtime of 30ms, which we consider as real-time capable in this work.

in Fig. 4, with runtimes above 80ms, the dense optical flow algorithms are too slow to be used in real-time applications

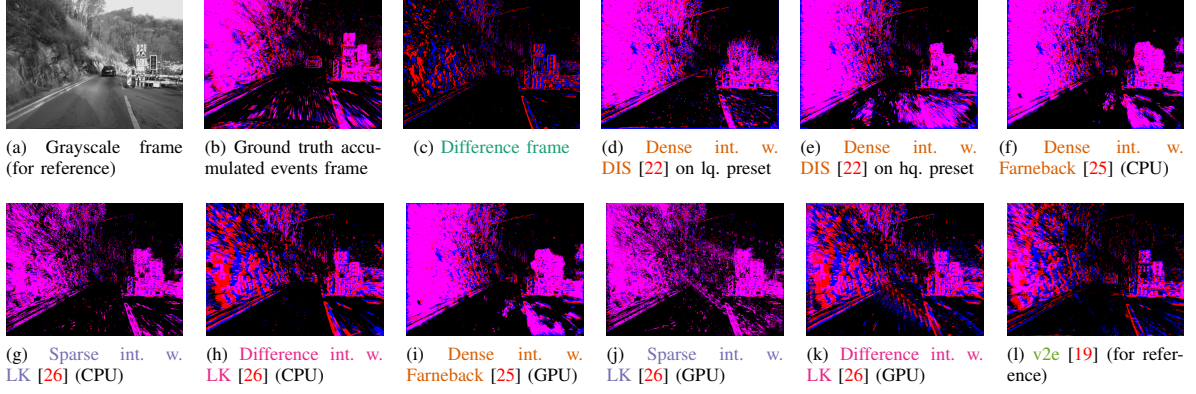


Fig. 5. Qualitative comparison of the output of the different simulation methods on a high events-per-frame video sequence depicting a car driving through Interlaken (composed from the DSEC [23] dataset). LK stands for Lucas Kanade [26]. The accumulation time was set to mimic the frame rate of the frame camera (1/fps).

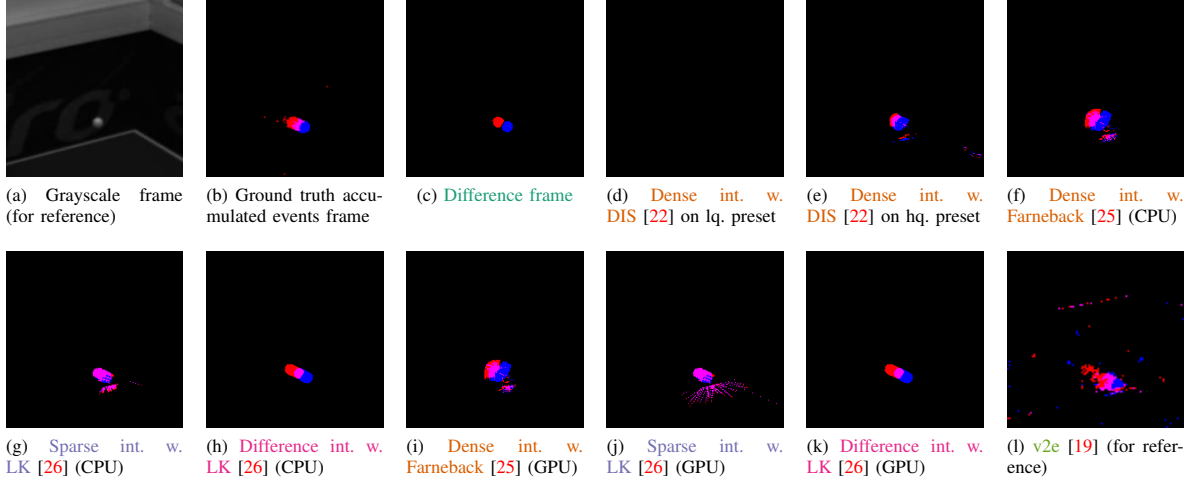


Fig. 6. Qualitative comparison of the output of the different simulation methods on a low events-per-frame video sequence depicting a flying table tennis ball. LK stands for Lucas Kanade [26]. The accumulation time was set to mimic the frame rate of the frame camera (1/fps).

in such scenarios. However, the dense method with DIS with the high quality preset is still three orders of magnitude faster than vid2e [18] and v2e [19]. Since there is only one moving ball in front of a static background in this dataset, the sparse interpolation method only needs to estimate the motion of the pixels representing the ball. As expected, the sparse interpolation method performs very well. The difference interpolation method, with its strict assumption met on this dataset, achieves the best runtimes.

B. QUALITATIVE COMPARISON

For qualitative comparison, the outputs of the different methods are shown in Fig. 5 for the dynamic dataset and in Fig. 6 for the less dynamic dataset. We used the same datasets as in Section IV-A. The accumulation time was set to mimic the frame rate of the frame camera (1/fps).

Discussion Starting with the dynamic dataset, the output

of the dense interpolation methods, shown in Figs. 5d–5f and 5i, bear a very strong resemblance to the ground truth accumulated event frame, shown in Fig. 5b. The positive and negative overlapping events, missing in the *difference frame* (Fig. 5c), are primarily generated from the interpolation. The impact of the different optical flow algorithms is relatively small, with the main difference being the amount of motion registered in the street area of the frames. The output of the sparse interpolation methods, shown in Figs. 5g and 5j, also bear a very strong resemblance to the ground truth accumulated event frame, shown in Fig. 5b. The output is noticeably noisier than the one of the dense interpolation methods, which is to be expected because of the sparse interpolation. If neighboring pixels are just under the threshold, they will not be interpolated and therefore do not emit an event. Additionally, some artifacts appear when Lucas Kanade [26] is run on the GPU. These artifacts are probably

a side product because only patches of the original image are sent to the GPU by OpenCV. The difference interpolation method performs quite poorly on this dataset, as can be seen in Figs. 5h and 5k. This is mainly due to the constraints not being met. The perspective of the objects in the scene changes quite drastically as the car is driving by them. However, some events between the two frames are still emitted, and the output is a clear improvement over the simple *difference frame*. The output of v2e comes closest to the one of the difference interpolation method from a visual point of view.

In the less dynamic dataset, the dense interpolation method does not perform as well, shown in Figs. 6d–6f and 6i. The dense interpolation method with DIS with the low-quality preset cannot even detect any motion, making the ball completely invisible, as seen in Fig. 6d. With the Farneback optical flow algorithm, the dense interpolation method manages to estimate some motion in the ball but creates an event cloud around it (Figs. 6f and 6i). Only the dense interpolation method with DIS on the high-quality preset produces a favorable result, as seen in Fig. 6e. These inaccuracies can be attributed to the dense optical flow algorithms. Traditional dense optical flow algorithms find the frame’s general motion in patches. This is especially a problem for DIS [22] with the low quality preset as the ball is not even registered because of the large patch size and the fact that the surrounding pixels do not move with the ball. The output of the sparse interpolation method has some artifacts due to the interpolation of overlapping subjects. One object represents where the ball has moved in the current frame, and one object where the ball was in the previous frame. The GPU implementation has some additional artifacts similar to the ones in the dynamic dataset. The output of the difference interpolation, shown in Figs. 6h and 6k, makes it apparent that this method is well suited for this kind of scene. It is the only algorithm able to correctly interpolate the ball’s position and create the events that lie between the two frames. As seen in Fig. 6l, v2e produces quite some artifacts and does not look as similar to the ground truth compared to most of our methods.

C. STATISTICAL COMPARISON

In this section, we present a statistical comparison of our event simulation methods with real events and the output of vid2e [18] and v2e [19]. We use the metric of events per pixel per second ($\frac{\text{events}}{\text{pixel} \cdot \text{s}}$) introduced in [27] as comparison for two short sequences. Event count is a necessary but not sufficient condition for a faithful simulation of an event camera. We took 2s (40 frames) from the *interlaken.00.c* sequence from [23] for high events-per-frame data and 1.13s (170 frames) from our flying table tennis ball dataset for low events-per-frame data. We adjusted the threshold values C_{pos} and C_{neg} for every simulation method to get similar mean values as the ones from the real events. We list the mean and standard deviation as well as the used threshold values C_{pos} and C_{neg} (unless unknown) in Table II for the high events-per-frame data and in Table III for the low events-per-frame

data.

Discussion As can be seen in Table II, for the high events-per-frame dataset, our simulation methods tend to have a bit of a lower standard deviation compared to the real events, v2e and vid2e. On the low events-per-frame dataset, however, most of our simulation methods have standard deviations closer to the one of the real events compared to v2e and vid2e.

TABLE II
EVENT STATISTICS ON A HIGH EVENTS-PER-FRAME VIDEO SEQUENCE

Method	Events / (pixel · s)		C_{pos}	C_{neg}
	mean	std. dev.		
Real events	62.7933	82.7128	?	?
Dense int. w. DIS, lq. preset	56.3079	58.9868	3	-3
Dense int. w. DIS, hq. preset	65.6304	66.793	3	-3
Dense int. w. Farneback	65.999	67.4279	3	-3
Sparse int. w. LK	59.3083	52.3827	6	-6
Difference int. w. LK	59.3034	55.0075	19	-19
v2e [19]	58.8313	78.3513	0.1	0.1
vid2e [18]	63.6588	77.9866	0.14	0.14

TABLE III
EVENT STATISTICS ON A LOW EVENTS-PER-FRAME VIDEO SEQUENCE

Method	Events / (pixel · s)		C_{pos}	C_{neg}
	mean	std. dev.		
Real events	0.046	1.0228	43	-17
Dense int. w. DIS, lq. preset	0.0472	1.165	13	-13
Dense int. w. DIS, hq. preset	0.0462	0.8783	35	-35
Dense int. w. Farneback	0.0445	0.574	19	-19
Sparse int. w. LK	0.0467	0.439	27	-27
Difference int. w. LK	0.0461	0.8785	112	-112
v2e [19]	0.0455	0.4912	0.575	0.575
vid2e [18]	0.0318	1.776	5.7	-5.7

V. CONCLUSION

This work presented multiple event simulator methods that can run in under 30ms. Compared to existing event simulators, ours can be used in real-time robotics applications while remaining competitive in the quality assessment. We have shown the influence of the resolution and the dynamics of the scene on the runtime of the simulators, the quality of the simulated events, and the statistics of the events.

Depending on the camera setup and the dynamics of the scene, we have the following suggestions: For scenes with high dynamics, e.g., an event camera mounted on a driving car, we recommend the dense interpolation method with Farneback on the GPU. In scenes with fewer dynamics, e.g., tracking an object with a static event camera, we recommend using the difference interpolation method. For use cases in between, some experiments might be needed to find the sweet spot between a fast runtime and a high quality of the simulated events. The sparse interpolation method might be a good choice.

With this work, we have made a first step towards event simulators for real-time robotics applications. We hope our work enables others to develop new algorithms for event-based data in real-time use cases.

REFERENCES

- [1] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–180, 2022.
- [2] A. I. Maqueda, A. Loquercio, G. Gallego, N. García, and D. Scaramuzza, "Event-based vision meets deep learning on steering prediction for self-driving cars," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5419–5427.
- [3] E. Perot, P. de Tournemire, D. Nitti, J. Masci, and A. Sironi, "Learning to detect objects with a 1 megapixel event camera," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 16639–16652. [Online]. Available: <https://proceedings.neurips.cc/paper/2020/file/c213877427b46fa96cfff6c39e837ccee-Paper.pdf>
- [4] I. Alonso and A. C. Murillo, "Ev-segnet: Semantic segmentation for event-based cameras," in *IEEE International Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019.
- [5] Y. Zhou, G. Gallego, X. Lu, S. Liu, and S. Shen, "Event-based motion segmentation with spatio-temporal graph cuts," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2021.
- [6] L. Pan, R. Hartley, C. Scheerlinck, M. Liu, X. Yu, and Y. Dai, "High frame rate video reconstruction based on an event camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.
- [7] M. Muglikar, D. P. Moeys, and D. Scaramuzza, "Event guided depth sensing," in *2021 International Conference on 3D Vision (3DV)*, 2021, pp. 385–393.
- [8] S. Tulyakov, D. Gehrig, S. Georgoulis, J. Erbach, M. Gehrig, Y. Li, and D. Scaramuzza, "Time lens: Event-based video frame interpolation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 16 155–16 164.
- [9] R. S. Dimitrova, M. Gehrig, D. Brescianini, and D. Scaramuzza, "Towards low-latency high-bandwidth control of quadrotors using event cameras," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2020. [Online]. Available: <https://doi.org/10.1109/icra40945.2020.9197530>
- [10] A. Vitale, A. Renner, C. Nauer, D. Scaramuzza, and Y. Sandamirskaya, "Event-driven vision and control for UAVs on a neuromorphic chip," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2021. [Online]. Available: <https://doi.org/10.1109/icra48506.2021.9560881>
- [11] J. Kaiser, J. C. V. Tieck, C. Hubschneider, P. Wolf, M. Weber, M. Hoff, A. Friedrich, K. Wojtasik, A. Roennau, R. Kohlhaas, R. Dillmann, and J. M. Zollner, "Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks," in *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. IEEE, Dec. 2016. [Online]. Available: <https://doi.org/10.1109/simpar.2016.7862386>
- [12] J. Nehvi, V. Golyanik, F. Mueller, H.-P. Seidel, M. Elgharib, and C. Theobalt, "Differentiable event stream simulator for non-rigid 3d tracking," in *CVPR Workshop on Event-based Vision*, 2021.
- [13] 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," *The International Journal of Robotics Research*, vol. 36, no. 2, pp. 142–149, Feb. 2017. [Online]. Available: <https://doi.org/10.1177/0278364917691115>
- [14] W. Li, S. Saeedi, J. McCormac, R. Clark, D. Tzoumanikas, Q. Ye, Y. Huang, R. Tang, and S. Leutenegger, "Interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset," in *British Machine Vision Conference (BMVC)*, 2018.
- [15] A. Radomski, A. Georgiou, T. Debrunner, C. Li, L. Longinotti, M. Seo, M. Kwak, C.-W. Shin, P. K. J. Park, H. E. Ryu, and K. Eng, "Enhanced frame and event-based simulator and event-based video interpolation network," 2021. [Online]. Available: <https://arxiv.org/abs/2112.09379>
- [16] H. Rebecq, D. Gehrig, and D. Scaramuzza, "ESIM: an open event camera simulator," *Conf. on Robotics Learning (CoRL)*, Oct. 2018.
- [17] D. Gehrig, M. Gehrig, J. Hidalgo-Carrio, and D. Scaramuzza, "Video to events: Recycling video datasets for event cameras," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2020. [Online]. Available: <https://doi.org/10.1109/cvpr42600.2020.00364>
- [18] Y. Hu, S. C. Liu, and T. Delbruck, "v2e: From video frames to realistic DVS events," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2021. [Online]. Available: <http://arxiv.org/abs/2006.07722>
- [19] D. Joubert, A. Marcireau, N. Ralph, A. Jolley, A. van Schaik, and G. Cohen, "Event camera simulator improvements via characterized parameters," *Frontiers in Neuroscience*, vol. 15, Jul. 2021. [Online]. Available: <https://doi.org/10.3389/fnins.2021.702765>
- [20] A. Z. Zhu, Z. Wang, K. Khant, and K. Daniilidis, "Eventgan: Leveraging large scale image datasets for event cameras," 2019. [Online]. Available: <https://arxiv.org/abs/1912.01584>
- [21] T. Kroeger, R. Timofte, D. Dai, and L. V. Gool, "Fast optical flow using dense inverse search," in *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 471–488. [Online]. Available: https://doi.org/10.1007/978-3-319-46493-0_29
- [22] M. Gehrig, W. Aarents, D. Gehrig, and D. Scaramuzza, "DSEC: A stereo event camera dataset for driving scenarios," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4947–4954, Jul. 2021. [Online]. Available: <https://doi.org/10.1109/ra.2021.3068942>
- [23] T. Finateu, A. Niwa, D. Matolin, K. Tsuchimoto, A. Mascheroni, E. Reynaud, P. Mostafalu, F. Brady, L. Chotard, F. LeGoff, H. Takahashi, H. Wakabayashi, Y. Oike, and C. Posch, "5.10 a 1280x720 back-illuminated stacked temporal contrast event-based vision sensor with 4.86μm pixels, 1.066geps readout, programmable event-rate controller and compressive data-formatting pipeline," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*. IEEE, Feb. 2020. [Online]. Available: <https://doi.org/10.1109/isscc19947.2020.9063149>
- [24] G. Farneback, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis*. Springer Berlin Heidelberg, 2003, pp. 363–370. [Online]. Available: https://doi.org/10.1007/3-540-45103-x_50
- [25] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision (darpa)," pp. 121 – 130, April 1981.
- [26] T. Stoffregen, C. Scheerlinck, D. Scaramuzza, T. Drummond, N. Barnes, L. Kleeman, and R. Mahoney, "Reducing the sim-to-real gap for event cameras," in *European Conference on Computer Vision (ECCV)*, august 2020.

Appendix B

eWand: A calibration framework for wide baseline camera systems

Authors

Thomas Gossard*, Andreas Ziegler*, Levin Kolmar, Jonas Tebbe and
Andreas Zell

Published in

2024 IEEE International Conference on Robotics and Automation
(ICRA 2024)

Time of publication

May 2024

DOI

10.1109/ICRA57147.2024.10610116

Included in this dissertation with permission from IEEE.
©2024 IEEE.

2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024, Yokohama, Japan

eWand: An extrinsic calibration framework for wide baseline frame-based and event-based camera systems

Thomas Gossard*, Andreas Ziegler*, Levin Kolmar, Jonas Tebbe and Andreas Zell

Abstract—Accurate calibration is crucial for using multiple cameras to triangulate the position of objects precisely. However, it is also a time-consuming process that needs to be repeated for every displacement of the cameras. The standard approach is to use a printed pattern with known geometry to estimate the intrinsic and extrinsic parameters of the cameras. The same idea can be applied to event-based cameras, though it requires extra work. By using frame reconstruction from events, a printed pattern can be detected. A blinking pattern can also be detected from the events. Such calibration methods can provide accurate intrinsic calibration for both frame- and event-based cameras. However, using 2D patterns has several limitations for multi-camera extrinsic calibration, with cameras possessing highly different points of view and a wide baseline. The 2D pattern can only be detected from one direction and needs to be of significant size to compensate for its distance to the camera. This makes the extrinsic calibration time-consuming and cumbersome. To overcome these limitations, we propose eWand, a new method that uses blinking LEDs inside opaque spheres instead of a printed or displayed pattern. Our method provides a faster, easier-to-use extrinsic calibration approach that maintains high accuracy for both event- and frame-based cameras.

SUPPLEMENTARY MATERIAL

The project’s code and additional resources are available at: <https://cogsys-tuebingen.github.io/ewand/>

I. INTRODUCTION

Before using cameras for robotic applications, one needs to calibrate them first. Camera calibration is the process of estimating the intrinsic camera parameters (the camera matrix $\mathbf{K}$ and the distortion coefficients $\mathbf{d}$) as well as the extrinsic camera parameters (the rotation $\mathbf{R}$ and translation $\mathbf{t}$ of the camera with respect to a reference frame) when multiple cameras are used. Thus, after calibrating a camera system, we know how a point in the 3D space will be projected into the 2D captured images. This enables the use of computer vision algorithms like triangulation or Perspective-n-Point.

Camera calibration for conventional cameras is a well-studied subject, with literature published over multiple decades [1][2][3]. The current best practice is to capture images of a printed pattern in various positions and orientations. The most popular patterns are the checkerboard, the asymmetrical circle grid, and AprilTags. Calibration with

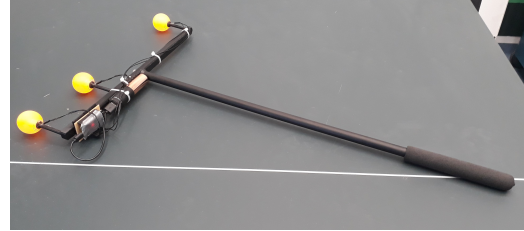


Fig. 1: Our eWand, composed of three of our markers.

such patterns work fine for a setup with a single camera or with cameras with a very similar field of view, e.g., stereo-vision setups. However, calibrating multiple cameras with different viewpoints can be problematic for two reasons. First, capturing images where the pattern is recognized by all cameras is not always possible, requiring calibration of cameras by pairs. When optimizing camera pairs individually, the pairwise optimal solution often does not coincide with the optimal solution using all cameras combined. Second, the pattern often has to be significantly tilted to be visible by multiple cameras, leading to difficulties in pattern detection and might lower the accuracy of the marker localization, potentially resulting in an inaccurate calibration. This is the reason why for such camera setups, 3D markers (such as spheres) are preferred instead of 2D markers. Sphere markers are always identically viewed, independent of the point of view. Commercial motion capture tools such as Optitrack¹ or Vicon² use “wands” to calibrate their cameras. Their infrared (IR) cameras capture the positions of IR-reflective spheres mounted on a stick. Though both Optitrack and Vicon are the industrial standard in terms of accuracy for motion capture, these systems are also quite expensive. Moreover, they require extra steps if they are to be used to calibrate a setup with standard RGB cameras. This concept has been translated to RGB cameras, using colored spheres and color filters to extract the position of the spheres [4]. However, such wands cannot be directly used for event-based cameras for calibration.

Event-based cameras are a new type of camera that have known rapid growth and increase in popularity in recent years. Instead of synchronously capturing all pixel values at periodic intervals, like frame-based cameras, event-based cameras continuously and asynchronously emit a signal (so-called *events*) on a log-intensity change above a threshold

* Equal contribution

The authors are with the Cognitive Systems Group, Dept. Informatics, University of Tuebingen. Corresponding author thomas.gossard@uni-tuebingen.de.

This work was funded by Sony AI.

¹<https://optitrack.com/>

²<https://www.vicon.com/>

for every pixel independently. Because of their low latency, high dynamic range, and non-redundant information flow, they offer many benefits for robotic applications [5]. They have already been used for stereo-vision [6], SLAM [7] and feature tracking [8].

As with frame-based cameras, event-based cameras also need to be calibrated first. The same calibration techniques as for conventional cameras cannot be directly applied. There are two approaches for event-based camera calibration: either frames are reconstructed from the events and then traditional calibration methods can be used [9] or the events are directly used [10][11].

Performing extrinsic calibration for frame-based and event-based cameras *jointly* requires having the pattern be detectable by both types of cameras *simultaneously*. The printed pattern and the LCD screen are the common approaches. The printed pattern is directly observable by the frame-based cameras and needs to be reconstructed for the event-based cameras. Depending on the camera used, the LCD screen can be set to either blinking mode or not. However, both also rely on 2D markers, which makes them inappropriate for a setup with multiple cameras with very different points of view. Moreover, for camera systems with a wide baseline between cameras, the pattern needs to be of significant size to be robustly detected from all cameras. It is obvious that a large LCD screen is heavy and difficult to move around. But this is also the case for printed patterns. The pattern's flatness requirement necessitates a rigid and consequently heavy support to prevent bending. This makes moving the pattern around difficult and the extrinsic calibration process slow. Wand-based calibration approaches are therefore preferred for multi-camera setups with very different points of view and wide baselines.

In order to make the wand detected by both, the frame-based and event-based cameras, we propose the use of blinking LEDs as markers. However, it isn't easy to pinpoint the center of an LED. Their small size and non-spherical shape make their detection and center regression difficult. Additionally, due to the high luminosity of the LED compared to the rest of the scene, smear, or blooming might be observed with the frame-based cameras, which makes it further challenging to estimate the center of the LED. Therefore, LED patterns tend to have a higher reprojection error [9]. To overcome these shortcomings, we propose placing these LEDs inside colored, opaque spheres. The opaque spheres act as a diffuser and are also easier to detect due to their size and symmetry. This enables both event and frame-based cameras to detect the balls. We name our method **eWand**. Using **eWand**, we can capture points over all cameras and use them to calculate the cameras' extrinsic parameters.

Contributions of this work are as follows:

- Robust and accurate markers for frame- and event-based cameras
- A detection method for said markers
- Extending the wand-based extrinsic calibration to event-based cameras

II. RELATED WORK

Since calibration of conventional cameras with printed calibration targets is a well-established topic with several toolboxes freely and commercially available [3][2][12], we will focus on the calibration of event-based cameras and wand-based calibration methods in the related work.

A. Event-camera calibration

As already mentioned, there are two methodologies for event-based camera calibration: Either frames are reconstructed from the event stream, and then traditional calibration methods can be used, or the events are directly used for the calibration. The first approach was introduced by [9]. In this work, E2VID [13] is used to reconstruct frames from the event stream. The reconstruction is a quite computationally heavy step (max frame rate of 11Hz for a resolution of 1280x720 with a NVIDIA RTX 2080 Ti GPU [13]). More recent reconstruction networks such as FireNet[14] provide faster reconstruction, being approximately three times faster than E2VID. However, even with reconstructed frames, pattern detection can be challenging. Even more so with a tilted pattern, which is often the case for patterns observed from highly different points of view. With the reconstructed frames, the kalibr calibration toolbox [3] is used to calibrate in the same way as for conventional cameras.

The second approach uses the events directly without any reconstruction for faster calibration and higher detection rates of the patterns. Early works make use of active markers such as blinking LED patterns [15][16] or blinking screens [17][18][19] to detect and extract the calibration patterns. In a more recent work [10], the authors introduced a dynamic event-based camera calibration algorithm. The approach directly calibrates given events captured during relative motion between the camera and the calibration pattern. However, the approach is not designed to calibrate multiple event-based cameras extrinsically. Although the calibration approach presented in [10] does use a circular pattern, often used for calibration of conventional cameras, it is not straight forward to calibrate event-based and frame-based cameras simultaneously. Another intrinsic calibration method was later introduced in [11]. Their approach is similar to [10] with improvements such as better clustering performance and achieving sub-pixel localization accuracy of the circular pattern.

Most of these methods could also be used for joint frame- and event-based camera calibration, if the event-based pattern extraction outputs the pattern in a data structure which can later be optimized together with the pattern extracted from the frame-based cameras. Reconstruction-based methods already rely on a pattern that can be directly used by the frame-based cameras. But as we explained, printed 2D patterns are not suited for calibrating multiple cameras with highly different points of view. Printed patterns are moreover not as straightforward as methods based on active markers. They can be set to always on for frame-based cameras or to blinking for event-based cameras. But the pattern then needs

to be perfectly static between each capture, which makes the calibration much more time-consuming and needs more care.

B. Wand-based calibration

While the majority of the camera calibration literature and available calibration tool boxes use printed calibration targets for their ease of use, there also exist methods that use a wand as calibration target [4][20]. Using a wand as a calibration target allows rapid, flexible, and accurate calibration of multiple frame-based camera systems and does not require the cameras to all face in the same direction or to have overlapping fields of view [4]. They can be used to calibrate extrinsics and the intrinsics to some degree, as shown by the Optitrack or Vicon systems (it is, however, unclear how the focal length of the cameras is estimated). While these methods were successfully applied for motion tracking with multiple infrared cameras, a wand-based calibration has never been tested to calibrate frame and event-based cameras jointly.

III. EVENT CAMERA MARKER

As previously mentioned, calibration targets are crucial for the calibration of cameras. In this section, we will describe our marker, which can be detected by frame- and event-based cameras.

Blinking LEDs have already been tested and used as markers for event cameras [9][21]. However, due to their shape, the identified position of the LED will depend on the point of view. Moreover, their small size makes detecting them from long distances difficult, as they can be mistaken for noise. We solve both of these issues by placing the LEDs in a spherical diffuser. The spherical shape of the diffuser makes the center of the marker independent of the point of view. This enables us to find the center of the marker more accurately. To ensure homogeneous and powerful lighting inside the diffuser, we used 2 high-intensity LEDs (7500MCD) oriented orthogonally. We tested the detection of our marker for different ambient luminosity, shown in fig. 2. The detection rate is the ratio of successful wand detections to the total number of observations. The marker

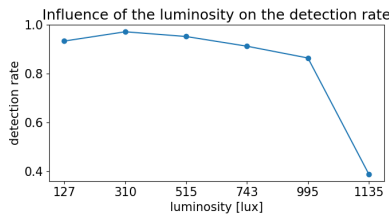


Fig. 2: Influence of the luminosity on the detection rate of the marker.

can be robustly detected for illuminations up to 1000 lux, which is the standard illumination of an office.

The blinking frequency of the blinking LED is controlled with a pulse-width modulation (PWM) signal from a micro-

controller. The duty cycle of the PWM signal was set to 50% to evenly space the on/off events for the event camera.

As for the blinking frequency of the LEDs, we tuned it to obtain the best detection rate with the event cameras' default settings. Intuitively, we want the highest possible frequencies in order to distinguish events generated from the blinking (high frequency) and events generated from the marker's movement (low frequency). However, because of the refractory period of event-based cameras, the blinking frequency cannot reach the kHz levels achievable by the microcontroller. In fig. 3, we show that frequencies higher than 600 Hz lead to lower detection rates of the markers.

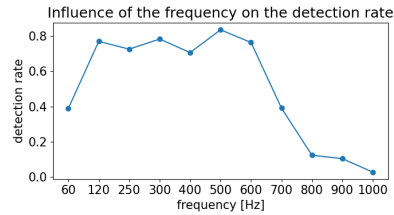


Fig. 3: Influence of the frequency on the detection rate of the marker.

A. Marker position extraction

In this section, we will describe the extraction of the marker position for frame-based cameras in section III-A.1 and for event-based cameras in section III-A.2.

1) *Frame-based camera*: For frame-based cameras, we identify the markers using a combination of color segmentation and blob detection, using the *OpenCV* implementation. However, false positives can still occur. To further filter out the detected blobs, we only keep blobs close to each other and aligned. An example of a simultaneously detected wand in each of the four frame-based cameras is visualized in fig. 4. It is important that the camera's exposure time is longer than a blinking period to avoid brightness inconsistencies in the marker's color. We also keep the exposure as low as possible to make the blinking marker stand out compared to the darker background.

2) *Event-based camera*: By tuning the biases of the event cameras (high and low pass filter mostly), it is possible to filter out most of the events except those generated by the marker.

However, this approach is not appropriate for tasks that also involve capturing events generated by other movements. Instead, we chose to extract the frequency of the generated events to filter out any frequencies that are not that of the blinking LEDs.

We rely on FrequencyCam [22] for that purpose. FrequencyCam takes an event stream as input and determines the frequency of every pixel in the field of view of the event-based camera. A visualization of the output of FrequencyCam can be seen in fig. 5b. We modified FrequencyCam to record only pixels corresponding to the frequency of our blinking LEDs,

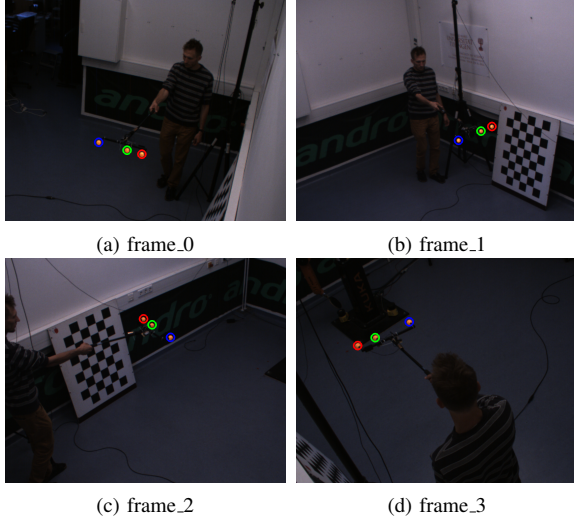


Fig. 4: Wand observed with the frame-based cameras. The position of the balls is marked with the red, green and blue circles.

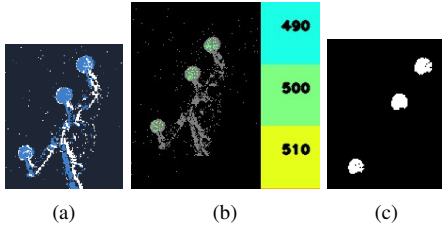


Fig. 5: (a) Accumulated events from the wand. (b) The detected frequencies of the marker positions by Frequency-Cam [22]. (c) Output of the blob detector.

employed blob detection (as shown in fig. 5c) for precise marker center determination, and exported both position data and timestamps.

IV. EWAND

In this section, we provide a general overview of our calibration method, visualized in fig. 6. We describe our calibration target in section IV-A. How we performed the intrinsic camera calibration is outlined in section IV-B. In section IV-C we detail the extrinsic calibration.

A. Calibration target

The wand is composed of three of our markers, described in section III, mounted in a line onto a stick, as shown in fig. 1. These spheres constitute the pattern's markers for the calibration. Marker size and spacing were chosen considering construction constraints and camera resolution. The spheres are aligned such that it is possible to distinguish them by the distance ratio between them. Indeed, the ratio is conserved regardless of the viewing angle. This is necessary to have marker correspondence between all cameras.

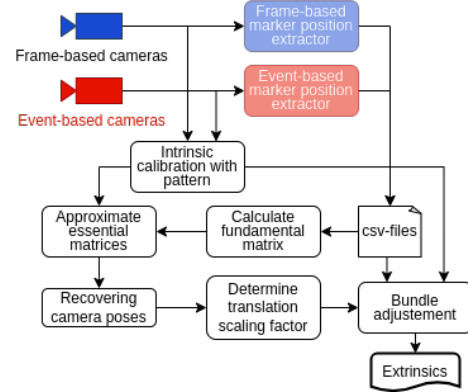


Fig. 6: An overview of our proposed calibration method.

The wand was 3D printed in one piece to ensure maximal accuracy of the markers' position. However, because it was hand-built, it is still prone to inaccuracies ($\pm 2mm$).

B. Intrinsic calibration

In order to estimate the camera matrix $\mathbf{K}$ and the distortion coefficients $\mathbf{d}$, we need a 3D or 2D pattern with known geometry. Multiple methods [23][24][25] are available, but for all of them, the 3D pattern requires more than three points. The intrinsic parameters' initialization method [26] requires planar patterns and cannot be applied to the wand, which is just a line.

Moreover, the inaccuracies in the marker positions of the wand, since it is self-manufactured, make it ill-suited for intrinsic calibration, which is very sensitive to bias and noise. For these reasons, we do not rely on the wand for the intrinsic calibration. We use the intrinsic parameters calculated with the *calibrateCamera* function from OpenCV with the asymmetric circle pattern and reconstruction with E2VID[13]. Camera intrinsics remain constant and only require one-time calibration, unlike extrinsics, which must be recalculated whenever a camera is moved even slightly. This is why the ease of use of eWand makes it preferable over other methods.

C. Extrinsic calibration

For the extrinsic calibration, we start by recording images and events while moving the wand around. After capturing the data from the event- and frame-based cameras, we extract the position of the markers, explained in section III-A.

We then rely on bundle adjustment for extrinsic calibration. It can be seen as an optimization problem that requires finding the best extrinsics and 3D marker positions to minimize the reprojection errors

$$\underset{\mathbf{P}_j, \hat{\mathbf{x}}_i, \mathbf{K}, \mathbf{d}}{\operatorname{argmin}} \sum_{i=1}^n \sum_{j=1}^m (x_{i,j} - \hat{x}_{i,j}(\mathbf{P}_j, \hat{\mathbf{x}}_i, \mathbf{K}, \mathbf{d}))^2 \quad (1)$$

where $x_{i,j}$ is the i th observed marker with camera j , $\hat{\mathbf{x}}_i$ is the estimated 3D position of the i th marker observation, $\mathbf{P}_j$ is the

projection matrix of the j^{th} camera, and $\hat{x}_{i,j}$ is the prediction of $x_{i,j}$ using $\mathbf{P}_j$, $\hat{\mathbf{X}}_i$ and the intrinsic camera parameters $\mathbf{K}$ and $\mathbf{d}$. The projection matrix is the product of the camera matrix, that we know, and of the camera pose, that we want to estimate. We rely on Ceres [27] to perform the optimization part of the bundle adjustment.

Additional costs are added to the optimization problem to use the extra information we have from the wands' structure as a constraint. We assume here that index 0, 1 and 2 belong to the same wand.

$$\text{Linearity: } (\hat{\mathbf{X}}_0 - \hat{\mathbf{X}}_1) \times (\hat{\mathbf{X}}_1 - \hat{\mathbf{X}}_2) = 0 \quad (2)$$

$$\text{Distance: } ||\hat{\mathbf{X}}_0 - \hat{\mathbf{X}}_1|| - l_{ref,0} + ||\hat{\mathbf{X}}_1 - \hat{\mathbf{X}}_2|| - l_{ref,1} = 0 \quad (3)$$

where $l_{ref,0}$ and $l_{ref,1}$ are the real distances between the wand's markers. Non-linear optimization problems, such as bundle adjustment, require a good initialization of the extrinsic parameters so as not to get stuck in the wrong local minimum. To do so, we first need the fundamental matrix $\mathbf{F}_j$ of a pair of cameras. The fundamental matrix is obtained using a RANSAC version of the 7-point algorithm. Once we have $\mathbf{F}_j$, we can calculate the essential matrix $\mathbf{E}_j$ with the camera matrix of the two cameras $\mathbf{K}_j$ and $\mathbf{K}_k$ using the formula: $\mathbf{E}_j = (\mathbf{K}_k)^T \mathbf{F}_j \mathbf{K}_j$. The essential matrix can then be decomposed into possible camera rotation $\mathbf{R}_j$ and translation $\mathbf{t}_j$, using SVD. However, the estimated translation vector $\mathbf{t}_j$ is only defined up to scale, but since we know the distance between the markers on the wand, we can retrieve the correct scale k using a golden-section search.

V. EXPERIMENT

A. Cameras

Our setup contains a total of four frame-based and two event-based cameras with baselines of 3m to 5m, organized as shown in fig. 7 (schematic is up to scale).

Four FLIR Chameleon3 frame-based cameras (1280×1024 pixels) are mounted in the room's corners. As for the event-based cameras, two Prophesee EVK4 cameras (1280×720 pixels) are placed at two corners of the room. The event-based cameras are "paired" with a frame-based camera to have both type of data with comparable points of view.

The parameters of both types of cameras were set to record the best quality of data. For the frame-based cameras, we want distinguishable ball outline compared to the rest of the scene. For the event-based cameras, we want to avoid noise and capture all the marker blinks. For that, we adjusted the trigger thresholds (defining the level of the illumination change necessary to trigger an event) to filter out noise but keep the events from the blinking markers.

B. Time synchronization

One of the most critical aspects of multi-camera calibration is the time synchronization of the data capture. Because the wand is constantly moving, a frame offset could lead to erroneous triangulation of the balls. In our setup, we use a microcontroller to trigger the frame-based cameras and the

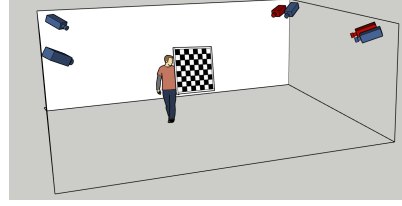


Fig. 7: Our camera setup consisting of four frame-based cameras (in blue) and two event-based cameras (in red) with baselines of 3m to 5m. Schematic is up to scale.

primary event-based camera with 50 Hz. The clocks of the primary and secondary event-based camera are synchronized via a synchronization cable between them.

C. Comparison

We compare the extrinsics estimated by our proposed calibration method, described in section V-F with two other calibration approaches. On the one hand, we use the popular calibration framework kalibr [3], described in section V-D. On the other hand, an intrinsic calibration method based on OpenCV [28] and an extrinsic calibration method using OpenCV and the optimization framework Ceres [27] to solve the calibration as a bundle adjustment problem, described in section V-E. In section V-G, we describe our evaluation of this experiment.

D. Calibration with kalibr

We performed the intrinsic and extrinsic calibration with one dataset for each calibration target, following [9].

E. Calibration via bundle adjustment with OpenCV/Ceres

For the calibration with OpenCV and bundle adjustment, we chose to perform the intrinsic and extrinsic calibration separately and on two different data sets.

Doing the calibration in two stages (first intrinsics and afterward extrinsics) has the advantages that recordings can be used which are specifically targeted for one of the two tasks. For the intrinsic calibration, the calibration target should be recorded in the cameras' whole field of view. On the other hand, for the extrinsic calibration, the calibration target has to be observed by all cameras simultaneously, not able to cover the individual cameras' whole field of view. We use two standard patterns: checkerboard and asymmetric circle grid. We chose not to include AprilTags as they tend to have a poor detection rate after reconstruction from events [9]. For the event-based cameras, E2VID [13] was used to generate reconstructed frames from the events.

a) *Intrinsic calibration:* For the intrinsic calibration, we used OpenCV functions to detect the calibration targets, followed by the *cameraCalibrate* function.

b) *Extrinsic calibration:* To perform the extrinsic calibration, we first calculated an initial estimate of the extrinsics and run a bundle adjustment afterward. For the initial estimate, we used *solvePnP*, given the observed target for every camera to get the position and rotation of the target

Camera	BA eWand (our)	kalibr (circleboard)	kalibr (checkerboard)	BA circleboard	BA checkerboard
frame_0	0.325 ± 0.257	0.274 ± 0.060	0.390 ± 0.325	0.218 ± 0.219	0.225 ± 0.204
frame_1	0.320 ± 0.271	0.311 ± 0.070	0.398 ± 0.335	0.201 ± 0.178	0.201 ± 0.185
frame_2	0.401 ± 0.361	0.168 ± 0.072	0.320 ± 0.320	0.178 ± 0.178	0.213 ± 0.193
frame_3	0.396 ± 0.389	0.159 ± 0.062	0.301 ± 0.243	0.199 ± 0.188	0.180 ± 0.165
event_0	0.603 ± 0.448	0.269 ± 0.068	0.513 ± 0.396	0.398 ± 0.365	0.459 ± 0.363
event_1	0.550 ± 0.431	0.295 ± 0.070	0.515 ± 0.405	0.469 ± 0.447	0.402 ± 0.370

TABLE I: Reprojection error (MAE) in pixels (mean and std.) for each calibration approach and camera, after the calibration. "BA" represents the bundle adjustment approach using OpenCV and Ceres, explained in section V-E.

in the camera frame. With these positions and rotation, we calculated the position and rotation of every camera with respect to the main camera. These initial estimates were then used as the starting point for a bundle adjustment problem, where we used Ceres [27] to optimize the intrinsics and extrinsics of all the cameras.

F. Calibration with the wand

We recorded a data set where we moved the wand around, covering the whole working space. The blinking frequency of the LEDs were set to 500 Hz. Since the wand is easier to detect, we were able to cover a bigger space compared to the other calibration targets. For the calibration we proceeded as explained in section IV.

G. Evaluation

For the evaluation we used two metrics: the commonly used reprojection error and a 3D position error.

1) *Reprojection errors*: Table I lists the reprojection errors obtained after calibrating with each method.

It is important to note that the measurements used differ for eWand and the other methods.

As can be seen, our method achieves similar reprojection errors as the other methods. Considering the standard deviation, the error ranges do overlap. The lower reprojection error of kalibr with the circleboard might come from the fact that kalibr does some outlier rejection, while our proposed method and OpenCV do not.

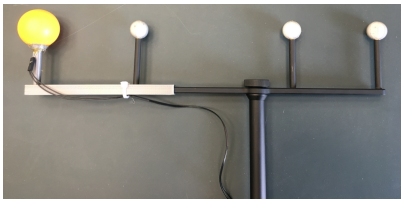


Fig. 8: Modified OptiTrack wand used for benchmarking.

2) *3D error*: For the evaluation, we attached one of our custom marker to an OptiTrack wand, as shown in fig. 8. The marker is the same as the one described in section III. We moved the OptiTrack wand with the attached marker within the workspace. Taking the 2D observations of the marker in all the cameras, we calculate the 3D position by solving a bundle adjustment problem, similar to eq. (1), where only the 3D position of the marker is optimized. Finally, we

compared the 3D position from the OptiTrack system, which we take as ground truth, to the ones we got from the bundle adjustment. The transformation between the Optitrack frame and our setup's frame is calculated by matching both generated point clouds. For the time synchronization of both, we determine the time offset by minimizing the error with the previously found transformation. As we can see in table II,

Method	3D positional error [mm] (mean and std)
BA (checkerboard)	47.0 ± 35.8
BA (circleboard)	84.9 ± 35.8
kalibr (checkerboard)	8.4 ± 10.5
kalibr (circleboard)	8.6 ± 10.4
BA eWand (our)	9.3 ± 10.4

TABLE II: The 3D position MAE [mm] (mean and std) of the different calibration methods.

the positional error of our proposed method (eWand) is very close to the results of kalibr. Considering the standard deviation, the error ranges do overlap. The bundle adjustment approach using OpenCV and Ceres performs a magnitude worse than kalibr and our approach. While kalibr does some outlier rejection, the bundle adjustment approach does not. The positional error with the checkerboard is the lowest in our experiments. One possible explanation is that on the frame reconstructions from the event data, the checkerboard detection might perform better compared to the circleboard.

VI. CONCLUSION

Calibrating camera systems is crucial, but the standard method of using printed patterns cannot be applied to event-based cameras. State-of-the-art methods include reconstructing frames or using active markers. Furthermore, calibration with the widely used calibration targets becomes challenging for multi-camera setups with a wide baseline. Our **eWand** method extends wand-based calibration to event-based cameras, using blinking LEDs to eliminate the need for frame reconstruction. **eWand** offers a simpler way to calibrate extrinsic parameters of multiple cameras, maintaining accuracy with less effort. While achieving good results in this first step, our bundle adjustment can be improved, e.g., by introducing outlier rejection. In the future, we also plan to make an improved wand with better manufacturing, for estimating intrinsic parameters as well. Overall, **eWand** aims to make setting up camera recording systems with frame- and event-based cameras more accessible and user-friendly.

REFERENCES

- [1] T. A. Clarke and J. G. Fryer, "The development of camera calibration methods and models," *The Photogrammetric Record*, vol. 16, no. 91, pp. 51–66, Apr. 1998. [Online]. Available: <https://doi.org/10.1111/0031-868x.00113>
- [2] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*. IEEE, 1999. [Online]. Available: <https://doi.org/10.1109/iccv.1999.791289>
- [3] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann, and R. Siegwart, "Extending kalibr: Calibrating the extrinsics of multiple IMUs and of individual axes," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2016. [Online]. Available: <https://doi.org/10.1109/icra.2016.7487628>
- [4] J. Mitchelson and A. Hilton, "Wand-based multiple camera studio calibration," *Center Vision, Speech and Signal Process*, 2003.
- [5] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–180, 2022.
- [6] Y. Zhou, G. Gallego, H. Rebecq, L. Kneip, H. Li, and D. Scaramuzza, "Semi-dense 3d reconstruction with a stereo event camera," in *Computer Vision – ECCV 2018*. Springer International Publishing, 2018, pp. 242–258. [Online]. Available: https://doi.org/10.1007/978-3-030-01246-5_15
- [7] J. Hidalgo-Carrio, G. Gallego, and D. Scaramuzza, "Event-aided direct sparse odometry," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2022. [Online]. Available: <https://doi.org/10.1109/cvpr52688.2022.00569>
- [8] P. Chibber, E. Perot, A. Sironi, and V. Lepetit, "Detecting stable keypoints from events through image gradient prediction," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2021. [Online]. Available: <https://doi.org/10.1109/cvprw53098.2021.00153>
- [9] M. Muglikar, M. Gehrig, D. Gehrig, and D. Scaramuzza, "How to calibrate your event camera," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2021. [Online]. Available: <https://doi.org/10.1109/cvprw53098.2021.00155>
- [10] K. Huang, Y. Wang, and L. Kneip, "Dynamic event camera calibration," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 7021–7028.
- [11] M. Salah, A. Ayyad, M. Humais, D. Gehrig, A. Abusafieh, L. Seneviratne, D. Scaramuzza, and Y. Zweiri, "E-calib: A fast, robust and accurate calibration toolbox for event cameras," 2023.
- [12] "Tvcial: Deploy multi-sensor calibration," <https://www.tangramvision.com/sdk/multimodal-calibration>, 2023.
- [13] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "High speed and high dynamic range video with an event camera," *IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI)*, 2019. [Online]. Available: http://rpg.ifi.uzh.ch/docs/TPAMI19_Rebecq.pdf
- [14] C. Scheerlinck, H. Rebecq, D. Gehrig, N. Barnes, R. Mahony, and D. Scaramuzza, "Fast image reconstruction with an event camera," in *IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, 2020, pp. 156–163.
- [15] "Calibration toolbox by rpg group," https://github.com/uzh-rpg/rpg-dvs_ros/tree/master/dvs_calibration, 2014.
- [16] M. J. Dominguez-Morales, A. Jimenez-Fernandez, G. Jimenez-Moreno, C. Conde, E. Cabello, and A. Linares-Barranco, "Bio-inspired stereo vision calibration for dynamic vision sensors," *IEEE Access*, vol. 7, pp. 138 415–138 425, 2019. [Online]. Available: <https://doi.org/10.1109/access.2019.2943160>
- [17] E. Mueggler, B. Huber, and D. Scaramuzza, "Event-based, 6-DOF pose tracking for high-speed maneuvers," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sept. 2014. [Online]. Available: <https://doi.org/10.1109/iros.2014.6942940>
- [18] "Calibration toolbox by vlo group," <https://github.com/VLOGroup/dvs-calibration>, 2017.
- [19] "Calibration toolbox by prophesee," https://docs.prophesee.ai/metavision_sdk/modules/calibration/guides/intrinsics.html, 2022.
- [20] K.-Y. Shin and J. H. Mun, "A multi-camera calibration method using a 3-axis frame and wand," *International Journal of Precision Engineering and Manufacturing*, vol. 13, no. 2, pp. 283–289, Feb. 2012. [Online]. Available: <https://doi.org/10.1007/s12541-012-0035-1>
- [21] M. LLC. (2023) Active marker 2d tracking sample - metavision sdk docs 4.3.0 documentation. [Online]. Available: https://docs.prophesee.ai/stable/samples/modules/cv/active_marker_2d_tracking_cpp.html
- [22] B. Pfrommer, "Frequency cam: Imaging periodic signals in real-time," 2022. [Online]. Available: <https://arxiv.org/abs/2211.00198>
- [23] R. I. Hartley, "Extraction of focal lengths from the fundamental matrix," 2001.
- [24] S. Bougnoux, "From projective to euclidean space under any practical situation, a criticism of self-calibration," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. Narosa Publishing House. [Online]. Available: <https://doi.org/10.1109/iccv.1998.710808>
- [25] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. USA: Cambridge University Press, 2003.
- [26] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000. [Online]. Available: <https://doi.org/10.1109/34.888718>
- [27] S. Agarwal, K. Mierle, and T. C. S. Team, "Ceres Solver," 3 2022. [Online]. Available: <https://github.com/ceres-solver/ceres-solver>
- [28] G. Bradski, "The OpenCV Library," *Dr. Dobbs's Journal of Software Tools*, 2000.

Appendix C

BiasBench: A reproducible benchmark for tuning the biases of event cameras

Authors

Andreas Ziegler*, David Joseph*, Thomas Gossard*, Emil Moldovan
and Andreas Zell

Published in

2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW 2025)

Time of publication

June 2025

DOI

Included in this dissertation with permission from IEEE.
©2025 IEEE.

BiasBench: A reproducible benchmark for tuning the biases of event cameras

Andreas Ziegler* Thomas Gossard* David Joseph* Emil Moldovan Andreas Zell
Cognitive Systems Group, University of Tübingen, Germany
{andreas.ziegler,david.joseph,thomas.gossard}@uni-tuebingen.de

Abstract

Event-based cameras are bio-inspired sensors that detect light changes asynchronously for each pixel. They are increasingly used in fields like computer vision and robotics because of several advantages over traditional frame-based cameras, such as high temporal resolution, low latency, and high dynamic range. As with any camera, the output’s quality depends on how well the camera’s settings, called biases for event-based cameras, are configured. While frame-based cameras have advanced automatic configuration algorithms, there are very few such tools for tuning these biases. A systematic testing framework would require observing the same scene with different biases, which is tricky since event cameras only generate events when there is movement. Event simulators exist, but since biases heavily depend on the electrical circuit and the pixel design, available simulators are not well suited for bias tuning. To allow reproducibility, we present BiasBench, a novel event dataset containing multiple scenes with settings sampled in a grid-like pattern. We present three different scenes, each with a quality metric of the downstream application. Additionally, we present a novel, RL-based method to facilitate online bias adjustments.

1. Introduction

The human eye operates differently from a traditional, frame-based camera. Its retinal receptors respond to changes in light within the environment, converting these changes into signals that are then transmitted to the brain. The eye can detect as many as a thousand light changes per second while ignoring the mostly static background. Event-based cameras, inspired by the human visual system, address several limitations of frame-based cameras, including low temporal resolution, high latency, limited dynamic range, and data redundancy. These bio-inspired sensors operate asynchronously, with each pixel independently detecting light changes and generating *events* that in-

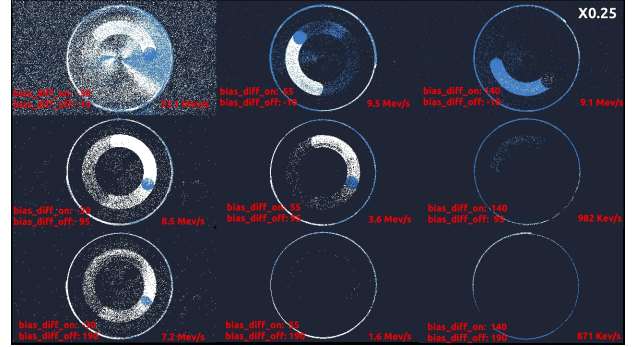


Figure 1. Accumulated event frames of the spinning disk setup of our event dataset with different *bias_diff_on* and *bias_diff_off*. As can be seen, different bias settings heavily influence the signal-to-noise ratio.

dicate whether luminosity has increased or decreased. This mechanism results in superior temporal resolution, allowing event-based cameras to capture fast movements with significantly less motion blur. Since only changes in light intensity are recorded, static parts of the image do not produce *events*. This makes event-based cameras very useful in many computer vision and robotics applications where a high temporal resolution, low latency, and a high dynamic range are needed.

Event-based cameras are configured through five parameters, commonly referred to as *biases*, which govern event detection thresholds and signal filtering. These five *biases* are the positive and negative thresholds for the event generation, the cut-off frequencies for a low- and high-pass filter, and the refractory period, which defines the dead time after a pixel generated an event, defined as the tuple $(bias_diff_on, bias_diff_off, bias_fo, bias_hpf, bias_refr)$. There is no universal bias configuration: they depend on the downstream task. Event-based cameras have been used for a wide range of applications, e.g., object tracking [22], image segmentation [2, 31], odometry [14]. They are particularly effective at capturing high-frequency phenomena [4], as well as scenes in both low-light conditions [29] and bright sunlight [8], provided the biases are appropriately se-

*equal contribution

This research was partially funded by Sony AI.

lected. This implies that the biases are tailored to the specific task. The full potential of event-based cameras can, therefore, only be realized when they are properly configured. A favorable choice of biases can result in a low noise and high contrast output, which is beneficial for the downstream application.

This raises the question of how to optimally control the biases of an event-based camera. In order to develop algorithms for bias tuning we need to be able to evaluate the effect of different settings. For this, event data from the same scene with different bias settings is needed. Furthermore, if we want to compare different algorithms, the event data should be reproducible. Recreating the same real-world recording setup in different research labs seems difficult to achieve. As for other applications, event simulators could be considered as an option. However, the *sim2real* gap could cause problems, especially since the biases depend very much on the electric circuit and the pixel design of the event camera, which are not accurately modeled in available event simulators. Therefore, we present a novel dataset that contains multiple scenes with a grid sampling of bias settings. A snapshot is shown in Fig. 1. Instead of using a real event camera to develop bias-tuning algorithms, our novel event dataset can be used. After the algorithm proposes a set of biases, the corresponding dataset recording can be used instead of changing the biases of an actual event camera. This allows a reproducible event stream without the need of a carefully designed calibration setup, speeding up the development of bias-tuning algorithms. Next to the event recordings with different sets of biases, we also provide a quality metric for the event streams of each of the three setups. Additionally, we introduce an initial baseline algorithm for tuning the two most commonly used biases, the on and off thresholds, leveraging our dataset to enable online bias optimization.

When tuning the biases, we are sequentially adjusting the biases to get the best event stream for our downstream application. This process can be formulated as a *cro:MDP* Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$, where:

- $\mathcal{S}$: observed event stream under the current bias settings.
- $\mathcal{A}$: the set of possible bias modifications that influence the event generation process.
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$: the probabilistic evolution of the event stream in response to bias changes.
- $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$: the reward for a given bias change.

However, both $\mathcal{P}$ and r are inaccessible in practice. Modeling $\mathcal{P}$ is infeasible for two reasons: (1) Event cameras capture brightness changes rather than absolute intensity, making it difficult to isolate bias effects from scene dynamics. (2) The event camera’s response to biases is highly

nonlinear and interdependent, lacking a precise physical model. Consequently, model-based *cro:RL* Reinforcement Learning (RL) is not applicable.

As for r , it could be defined via properties like event sparsity, noise reduction, or task performance. While *cro:ER* Event Rate (ER) can quantify sparsity, real-time noise estimation remains unavailable, and event stream quality depends on the task. To bypass the need for a cost function, we propose to instead use *cro:BC* Behavioral Cloning (BC) to adjust camera biases.

Contributions of this work are as follows:

- We present a novel event dataset for bias tuning, containing multiple scenes covering a grid of bias settings
- For each scene, we provide a quality metric and, therefore, a quality measure for each bias configuration
- We present a Behavioral Cloning (BC) pipeline as a baseline algorithm that is able to tune the two most common biases of event cameras online
- We make the recorded dataset, the metric, and the baseline algorithm publicly available

2. Related Work

We structure the related work as follows: First, we discuss bias settings and the best practices for manual tuning in Sec. 2.1. Then, we review approaches for automatic bias tuning in Sec. 2.2.

Finally, we cover key aspects of RL relevant to this study in Sec. 2.3.

2.1. Bias Settings

A first step towards automatic bias tuning is understanding the biases and their effect on the event stream. Several papers meticulously document the behavior of the event-based camera. In [11], the behavior of the DVS pixels for different bias parameters is described through extensive experimentation and measurements. In addition, the authors propose a new way to interpret events as a combination of signal, noise, and leakage. An outline of the manual bias optimization process is given, and general indications are proposed for different scenarios. This, and several other papers [10, 24], have described the process of manually finding optimal biases. The most widely used metric in the literature for rating an event camera’s output is the Event Rate (ER) [5, 21]. However, in cases where the event camera is used for object detection, another way of rating the output is by measuring its sharpness, a method proposed in [25]. This is achieved by calculating the average gradient of the frame. The authors conclude that this method shows promise but highlights the potential advantages of a metric that operates directly on the event stream without requiring frame generation.

In all the mentioned work, tuning the biases has to be done manually. Other works have simply used the default

settings of the camera [3] or do not mention them [9]. Our work provides the necessary dataset to develop algorithms that tune the biases automatically. In addition, we also present a first baseline algorithm.

2.2. Automatic Bias Tuning

Comparatively, little has been done in the field of automatic bias tuning, given the plethora of research around event-based computer vision. Multiple papers [10, 11, 21] dealing with the problem of finding optimal bias settings have resorted to measuring and rating the output for a range of different bias settings and then manually changing these biases.

In [5], the authors presented a closed-loop control method, representing a first step towards automatic bias tuning, using a combination of three fixed-step feedback controllers. These feedback controllers take measurements of the ER and noise and regulate them using all five bias parameters. The noise is calculated using a denoising algorithm and measuring how many events are removed. An optimal range is defined for the ER and the noise, and a simple algorithm changes the bias values if the two measurements leave their optimal ranges. The ER is regulated by increasing the thresholds if it is too high and decreasing them if it is too low. The ER is also limited by increasing the refractory period. The noise is controlled by adjusting the bandwidth. One limitation is that achieving the optimal bias setting requires multiple steps. Additionally, each threshold adjustment can trigger a burst of events, potentially leading to a short-term decline in the quality of the event stream with every change. Our proposed baseline does not require multiple steps, as it suggests the bias change necessary to reach an optimal bias in one step.

Another approach, and the most advanced work on the topic of which we are aware of, builds on the same idea but uses a two-part algorithm [21]. The algorithm controlling the biases uses two methods: fast adaptation, which changes the refractory period in order to limit the ER, and slow adaptation, which consists of changing the thresholds and adjusting the low-pass filter if fast adaptation is not enough to limit the ER. This split into two adaptations is necessary because only the refractory period does not cause visual artifacts when changed, which is problematic for the visual place recognition task presented in [21]. This means that the other three bias parameters should not be changed too often so as not to cause too much noise. For downstream applications such as visual(-inertial) odometry, SLAM, or object detection, this could lead to missing objects or losing track of the trajectory. While this approach is more advanced, it only considers the ER as a metric for the quality of the event stream. On the contrary, in our work, we rely on expert demonstrations so that the pipeline mimics an expert and does not only rely on the ER.

2.3. Reinforcement Learning

Instead of manually designing a control algorithm, RL has proven to be successful in learning complex control policies given enough demonstrations [17, 27]. In a more similar task, like adjusting the exposure time of a frame-based camera, RL was also successfully applied [20]. For RL algorithms to converge to something meaningful, a valid reward function is needed. This even led to its own research directions with reward engineering [28] and reward shaping [19].

Unfortunately, we do not have access to a metric capable of evaluating the quality of the events compared to frame-based cameras, which makes vanilla RL not applicable. As is evident from the literature on bias tuning for event-based cameras, there is no universal metric to measure the quality of an event stream. Many works use the ER as a quality indicator, but this is too short-sighted. This is applicable to certain tasks, but depending on the scene, the ER might vary greatly and is therefore not transferable to other problems. Instead of going the route of reward engineering, we rely on *cro:IL* Imitation Learning (IL), a subgroup of RL. The reward function is either learned from data, which is called *cro:IRL* Inverse Reinforcement Learning (IRL), or the behavior is learned directly from expert demonstrations in a supervised fashion, which is called Behavioral Cloning (BC).

In a first work [1], the authors introduced an apprenticeship learning algorithm that uses demonstrations from an expert to learn via Inverse Reinforcement Learning (IRL). IRL was further improved by employing the principle of maximum entropy in [32]. It is further extended by using a Fully Convolutional Neural Network (FCNN) for the reward instead of a linear mapping of features, which allows for more complex feature mappings [30]. Data-driven RL has also been used in robotics applications, e.g., to control a robot arm with images as input [28].

In this work, we developed an BC policy for bias tuning, which serves as a baseline. Specifically, we use TD3+BC [6], a simple approach to offline RL where TD3 [7] has been changed in two ways: A weighted behavior cloning loss is added to the policy update, and the states are normalized.

3. Dataset

As previously mentioned, to develop algorithms for automatic bias tuning, event data from the same scene with different bias settings is needed. Since there is no standard approach to tuning the biases on event cameras yet, there is also no standard setup to do so. While the number of possible scenes is unlimited, everyone could build their own setup. We believe reproducible data in the form of a benchmark can help the community to make comparisons.

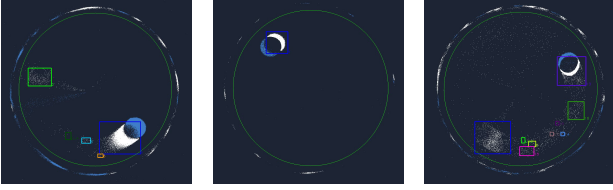


Figure 2. The spatter tracker from the Metavision SDK for different event data with the bias settings (left) (20, -10, -35, 24, 126), (middle) (20, 25, 10, 0, -20), and (right) (-10, 95, 10, 0, 53).

Although there are multiple event simulators available these days [15, 16, 23, 33], none of them model all the bias settings accurately, which is essential given that the biases are very much dependent on the electric circuit and pixel design of an event camera. Therefore, we present a dataset containing multiple setups with various bias settings, each with two scenes to be used for training/development and test. Changing the biases of the camera will be simulated by changing the file being read, so as to match the new biases. For each setup, we provide a quality metric and a quality measurement for each recorded bias configuration. This dataset is intended to serve the community as a benchmark and enable the development of bias-tuning algorithms.

All recordings were captured using Prophesee EVK4 event cameras. The spinning dot and blinking LED datasets were recorded with an 8mm lens, while the robot arm dataset was recorded with a 6mm lens. The focus was adjusted using the Metavision Focus Adjustment tool¹, and the aperture was set to $f/4$.

We recorded two scenes of the same setup with the same lighting conditions so that any differences between the recordings are caused by the bias settings and not by changes in the scene. To allow for the development of model-based or learning-based algorithms for bias tuning, we need setups for which we can define a quality metric. Thus, we built three different setups: a spinning dot, an LED board consisting of LEDs with different blinking frequencies and waveforms, and an event camera attached to a robot arm, performing a trajectory as an cro:VOVisual Odometry (VO) example.

3.1. Spinning Dot

The spinning dot has become a famous example to explain the working principles of an event camera. We brought this example alive by attaching a grey or black dot to a spinning disk.

We spin the dot at 10 rps and film it with the event camera 0.4m away from the dot spinner. The setup is shown in Fig. 11 in the supplementary material. Using the event camera, we captured 38,880 recordings, each with a grey

¹<https://docs.prophesee.ai/stable/samples/modules/calibration/focus.html>

Bias	Start	End	Number of biases
<i>bias_diff_on</i>	-30	130	18
<i>bias_diff_off</i>	-10	190	18
<i>bias_fo</i>	-35	55	5
<i>bias_hpf</i>	0	120	6
<i>bias_refr</i>	-20	200	4

Table 1. Bias values for the spinning dot setup. We choose an equidistant bias selection.

and a black dot, with different bias settings. Each recording is one second long. We also wait a bit after each bias update to ignore the event burst, which can occur when changing the bias parameters. The light source comprises LED panels without pulse modulation, resulting in a constant brightness at an experiment level of around 1500 lumen.

For the bias settings, we choose an equidistant selection covering the whole range of the biases, listed in Tab. 1. More details can be found in Sec. 8.1 in the supplementary material. To obtain a quality metric for the spinning dot, we used the spatter tracker from the Metavision SDK², shown in Fig. 2. The accumulation time was set to 1ms, and the tracking was limited to within the spinning disk to ignore events generated by the edge of the disk.

In this work, we introduce three distinct metrics for evaluating the quality of the tracking, taking into account the influence of the biases of event cameras. The first metric is the tracklet fragmentation TF , which quantifies the number of distinct tracklets corresponding to the spinning dot. Ideally, this value should be 1, indicating continuous tracking without interruption or fragmentation. The second metric is the tracking length TL , representing the total duration for which the dot is being tracked. This includes periods when the dot is tracked under different tracking identifiers (IDs). A longer tracking length suggests more consistent tracking, though varying tracklet IDs may indicate tracking challenges. Additionally, we record the total number of different tracklets $N_{tracklets}$ detected during the tracking process. In an ideal scenario, this number should be 1, reflecting that only the spinning dot is being tracked. Any additional tracklets are typically due to noise or false positives, such as detected spatters, which we aim to mitigate. We consider biases valid when $TL > 0.75$. Under this criterion, 44% of the settings result in successful dot tracking for the black dot.

3.2. Blinking LED Board

To see the influence of the low-pass and high-pass filter, controlled by the biases *bias_fo* and *bias_hpf*, we designed

²https://docs.prophesee.ai/stable/samples/modules/analytics/tracking_spatter_py.html#chapter-samples-analytics-tracking-spatter-python

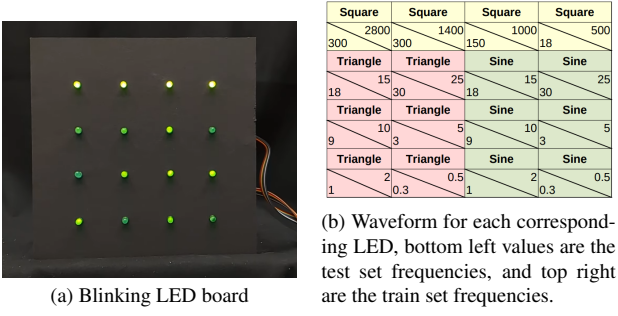


Figure 3. Blinking LED setup

Bias	Start	End	Number of biases
<i>bias_diff_on</i>	-80	120	11
<i>bias_diff_off</i>	-30	170	11
<i>bias_fo</i>	-29	48	8
<i>bias_hpf</i>	4	116	8
<i>bias_refr</i>	-15	225	4

Table 2. Bias values for the LED board setup. We choose an equidistant bias selection.

an LED board, shown in Fig. 3a. We used a 4×4 LED grid, each emitting different waveforms to introduce varying light intensity patterns, as visualized in Fig. 3b.

The four LEDs in the top row emitted a square waveform with a 0.5 duty cycle, each operating at a different frequency. The six LEDs on the lower left followed a sine wave with six distinct frequencies, while the six on the lower right emitted a triangular waveform with matching frequencies. This setup was designed to ensure diverse light intensity variations. We set the frequencies to different values to distinguish the train and test set. These frequencies are described in Figure 3b.

For the bias settings, we choose an equidistant selection covering the whole range of the biases, listed in Tab. 2. This gave us a total of 30,967 recordings. More details can be found in Sec. 8.2 in the supplementary material.

As a quality metric, we estimated the frequencies of the LEDs by fitting a cosine function to the difference in positive and negative events within the region associated with the LED. With this estimated frequency and its error, f_{est} and Δf_{est} respectively, the true frequency, f_0 , we define our metric as Relative Frequency Uncertainty (RFU)

$$RFU = \frac{|f_{est} - f_0| + \Delta f_{est}}{f}. \quad (1)$$

A value of 0 indicates a perfect frequency estimation with a negligible error, while higher values represent the relative size of misalignment and error compared to value. We cut this at a value of 2 to not overestimate on single bad-fitted frequencies. We consider biases valid when every fre-

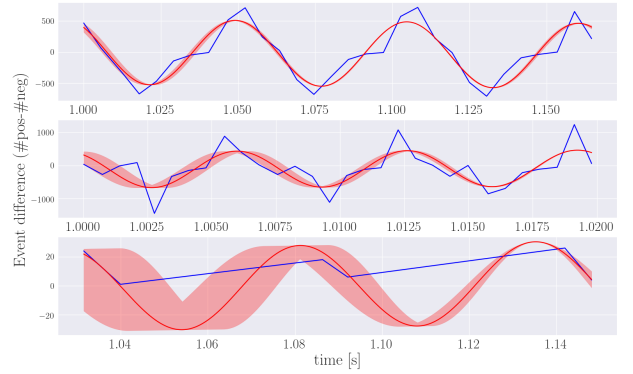


Figure 4. Examples of the frequency estimation for different LEDs and bias settings. The real event data is given in blue, the best fit in red. The red-shaded area resembles the uncertainty of the fit from the frequency estimate. The resulting Relative Frequency Uncertainty (RFU) is for the function on the top 0.018, for the function in the middle 0.04, and for the function on the bottom 0.4.

Bias	Start	End	Number of biases
<i>bias_diff_on</i>	-50	140	15
<i>bias_diff_off</i>	-10	190	15
<i>bias_fo</i>	-30	55	5
<i>bias_hpf</i>	0	120	6
<i>bias_refr</i>	0	0	1

Table 3. Bias values for the VO setup. We choose an equidistant bias selection.

quency could be estimated with an RFU of below 2. Under this criterion, 55% of the settings result in a successful frequency estimation.

3.3. Visual Odometry (VO)

For a more application-oriented setup, we recorded event data intended to be used for event-based VO pipelines. We recorded the events for a trajectory in which the camera moved left and right in one plane for 10s. To get a repeatable motion and an accurate ground truth trajectory, we mounted an event camera to the end-effector of a Pandas robot arm, shown in Fig. 18 in the supplementary material. The ground truth trajectory from the end-effector of the robot arm is shown in Fig. 5.

For the bias settings, we choose an equidistant selection, covering the whole range of the biases listed in Tab. 3. More details can be found in Sec. 8.3 in the supplementary material.

To get a metric, we run DEVO [18], a state-of-the-art event-based VO pipeline, on all event recordings and evaluated the resulting trajectories with evo [12]. This led to 6750 recorded event streams and trajectories.

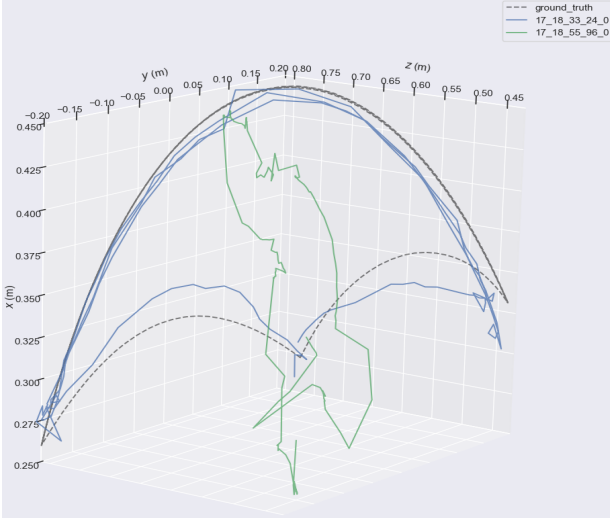


Figure 5. The ground truth trajectory of the end-effector of a Pandas robot arm is shown in black. The trajectory output from DEVO [18] for the *bias* settings (17, 18, 33, 24) is shown in blue. And the trajectory output for the *bias* settings (17, 18, 55, 96) is shown in green. Despite the similar bias values, the VO failed in one case.

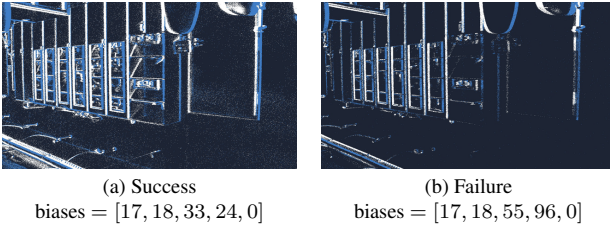


Figure 6. Comparison of accumulated event frames of (a) a successful and (b) a failed estimated trajectory with different biases.

In Figure 6, we present accumulated event frames for two different bias configurations. Although the event frames appear visually similar, DEVO successfully inferred the trajectory in one case but failed in the other, as shown in Fig. 5. This underscores the fact that even when the event stream looks comparable to human observers, its effectiveness in downstream tasks can vary significantly.

From all recorded trajectories, we visualize the cro:APEAbsolute Pose Error (APE) of five selected ones in Fig. 7. These trajectories are chosen such that their errors are evenly spaced across the range of observed errors.

From Fig. 7, it can be seen that only the recording with the bias settings $(-9, 4, -8, 0, 0)$ led to a reasonable APE. This shows the importance of bias tuning in order for a downstream application to give reasonable results.

We consider biases valid when $APE < 0.13$. Under this criterion, 10.05% of the settings result in a successful tra-

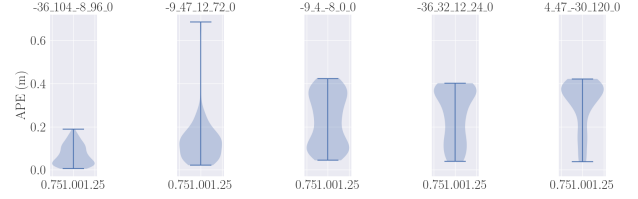


Figure 7. The Absolute Pose Error (APE) plot of five trajectories, each with different bias settings, estimated by DEVO [18]. These five trajectories are picked to have an equidistant error between each other.

jectory estimate.

4. A Baseline

In addition to the presented dataset and to showcase how our dataset can be used, we developed a BC based policy baseline to tune the two most commonly used biases, *bias_diff_on* and *bias_diff_off* with the other biases set to default values.

4.1. Method

As previously mentioned, we framed bias optimization as a MDP defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$. Since formulating a good reward function is complicated and there is not yet a cost function the literature agrees on, we choose an IL based approach. Therefore, we leverage expert demonstrations using our dataset and employ a BC algorithm to learn a policy for adjusting the two threshold biases, *bias_diff_on* and *bias_diff_off*. We used the recorded event streams from our novel dataset and manually generated expert demonstrations by specifying the biases we estimated were optimal.

The input of the model is a compressed latent space of the event stream that we get from a feature extraction pipeline and an expert demonstration. The feature extraction is done in multiple steps. We start by accumulating *ON*-events and *OFF*-events for an accumulation time of 8ms, each resulting in one channel. These two channels have the resolution of the used event camera (1280×720 pixels). Due to hot pixels, there are pixels with some events that are orders of magnitude higher than the rest of the pixels. This will lead to a decrease in sensitivity in the nondefect pixels after normalization. Therefore, we do not normalize the input by dividing with the maximal value. We linearly scaled the 90% of data with the lowest event count to 0.9 while mapping the remaining 10% to the range from 0.9 to 1. This way, we do not ignore hot pixels but do not let them dominate the input data.

The next step is applying a pre-trained ResNet [13] as a feature extraction algorithm on the normalized input. To conform with the input dimensions of ResNet, we added an empty third channel to the two accumulated event channels.

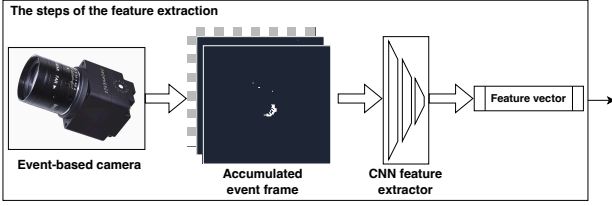


Figure 8. The steps of the feature extraction from the event stream to a vector of floats. We use ResNet50 [13] as a feature extractor.

ResNet uses an input resolution of 224×224 , whereas our accumulated event frames have a resolution of 1280×720 pixels. To resolve this incompatibility, we divide the histograms into sub-frames and apply the feature extraction on each of these sub-frames. For each subframe, we get a feature vector with a size of 1000. These stacked feature vectors can then be used as input for the BC learning algorithm. All the steps involved in the feature extraction are sketched in Fig. 8.

Next to the extracted features from the event data, the second input to the BC model is an expert demonstration. The demonstrations need to have the same format as the actions, so the expert must choose the absolute bias settings that seem best for each recording or change the biases needed to reach this setting. Other work [20] found that relative changes perform better than absolute values, so we have decided to also use relative bias changes as expert demonstrations. For the expert demonstration, we choose optimal bias settings where the ER is low, indicating a low level of noise, while the spinning dot is clearly visible.

For the BC algorithm we adapted the TD3+BC algorithm [6] from its implementation in the online library d3rlpy³ [26]. This adaptation includes the adjustment of the input to accept our features and calculating the loss using the Euclidean distance between the prediction and the expert demonstration. The overall pipeline used for this learning process is illustrated in Fig. 19 in the supplementary material.

For training, we created 10000 randomly distributed accumulated event frames. As a loss, we implemented the mean Euclidean distance between predicted and expert action. As hyperparameters, we chose a learning rate of 0.0003, a batch size of 265, a discount factor of 0.99, and Adam as the optimizer.

To evaluate the capabilities of the learned BC model, we performed the following experiments.

4.2. Experiments

Influence of the training and evaluation data Having two datasets with the spinning dot, one with a black dot and one with a gray dot, we investigated the influence of

Model trained on	Model validate on	Test loss
Black dot dataset	Grey dot dataset	0.26
Grey dot dataset	Black dot dataset	0.2

Table 4. The final loss values with different training and test data after the training process finished. The loss is given in the Mean Squared Error of the normalized proposed action.

the training and test dataset. We speculated that training the model on the recordings of the black dot would lead to a model that generalizes far worse than the one trained on the recordings of the grey dot because the black dot is always visible even for the maximum values of the thresholds as opposed to the grey dot. This means that the model would never encounter the state in which the dot is not visible in training, leading to it suggesting sub-optimal actions if it encounters that state. We tested this hypothesis by performing different runs. In one run, we trained the model on recordings of the black dot and validated it on the grey dot. In another run, we trained the model on the grey dot and validated it on the black dot. For the test dataset, we created 200 accumulated event frames from the 100 recordings of the other dot. The results of this comparison in Tab. 4 prove our hypothesis that the model trained on the black dot is less accurate.

Object Tracker We use the same metrics as defined in Section 3.1. We used three different initial bias pairs to test starting configurations. As inadequate bias values, we used $bias_diff_off = -10$ and $bias_diff_on = -35$. For sub-optimal bias values, we used $bias_diff_off = 65$ and $bias_diff_on = -10$. And as optimal bias values $bias_diff_off = 40$ and $bias_diff_on = 40$. After the bias values were set, we used our model to adjust the biases and then started the object tracker.

The results of this experiment are listed in Tab. 5, and an example of a change in bias is shown in Fig. 22. The model struggled with large initial $bias_diff_off$ values, proposing excessive bias changes that weakened the signal, but this is uncommon in real-world scenarios where default bias values as a starting point ensure stable bias adjustments.

Convergence Behavior In order to check whether the model converges for any bias value and does not simply maximize the thresholds for every scene, we run our model for each of the 100 grey dot recordings and report the sum of the relative bias changes of the threshold predictions. The result as a heat map can be seen in Fig. 9. The absolute value of the actions gets smaller and smaller until they reach the setting $bias_diff_off = 40$ and $bias_diff_on = 40$, where they are closest to zero. This indicates that the model correctly

³<https://github.com/takuseno/d3rlpy>

Initial biases	Average proposed bias change		Demonstration bias change		Success rate
	<i>diff_off</i>	<i>diff_on</i>	<i>diff_off</i>	<i>diff_on</i>	
-10, -35	21 ± 10	100 ± 10	25	75	1.0
65, -10	6 ± 5	35 ± 10	-10	10	0.8
40, 40	10 ± 5	16 ± 5	0	10	1.0

Table 5. Results of an object tracker experiment over 10 runs. The initial bias settings are expressed as two numbers which represent the settings *bias_diff_off* and *bias_diff_on*. The last column expresses the success rate of how many times the model has found an optimal bias setting and was able to track the dot over the number of experiments.

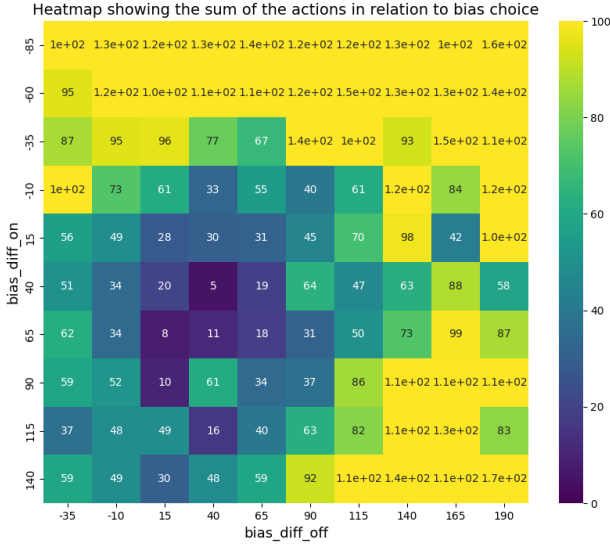


Figure 9. A heat map showing the sum of the relative bias changes suggested by the model for different *bias_diff_off* and *bias_diff_on* for the recordings of the grey dot. For the bias pair *bias_diff_off* = 40 and *bias_diff_on* = 40, the sum of the actions is the smallest, which resembles the optimal bias settings for this scene.

identifies this bias pair as an optimal bias setting. Additionally, the absolute value of the actions is larger the farther away the bias settings are from this setting.

The same thing happens for the recordings of the black dot. The corresponding heat map is shown in Fig. 10. We can see that the sum of the bias changes generally gets smaller the closer the biases are to the value of 40 for *bias_diff_off* and the value 65 for *bias_diff_on*. The model converges around these values, which are different compared to the convergence values for the grey dot. Therefore, the model recognizes the different scenes and has different predictions and optimal biases for each one.

5. Limitations and future work

Our approach is limited by the discrete and relatively large step sizes used during bias tuning, which may restrict fine-grained optimization. Additionally, the metrics employed

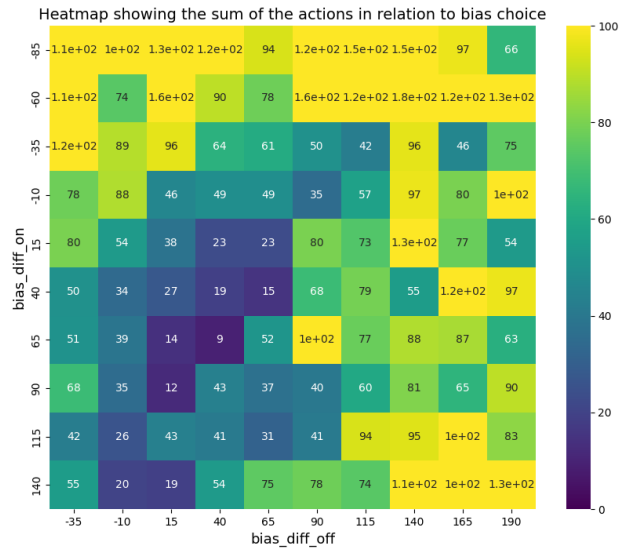


Figure 10. A heat map showing the relative bias changes suggested by the model for different *bias_diff_off* and *bias_diff_on* for the recordings of the black dot. For the bias pair *bias_diff_off* = 40 and *bias_diff_on* = 65, the sum of the actions is the smallest, which resembles the optimal bias settings for this scene.

are inherently biased by the parameters of the downstream application, potentially limiting their generalizability across different tasks.

6. Conclusion

Event-based cameras offer advantages in computer vision but require well-tuned biases for optimal performance. To facilitate automated bias tuning, we introduced *BiasBench*, a dataset of scenes captured under varying biases with corresponding quality metrics. This dataset enables the development and evaluation of tuning algorithms, fostering a better understanding of bias selection for different tasks. Additionally, we presented a baseline RL-based approach that learns bias tuning from expert demonstrations using Behavioral Cloning (BC).

References

- [1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Twenty-first international conference on Machine learning - ICML '04*, page 1. ACM Press, 2004. 3
- [2] Iñigo Alonso and Ana C Murillo. Ev-segnet: Semantic segmentation for event-based cameras. In *IEEE International Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2019. 1
- [3] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. A low power, fully event-based gesture recognition system. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. 3
- [4] S. Baldini, R. Bernardini, A. Fusiello, P. Gardonio, and R. Rinaldo. Measuring vibrations with event cameras. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVIII-2/W7-2024:9–16, 2024. 1
- [5] Tobi Delbruck, Rui Graca, and Marcin Paluch. Feedback control of event cameras. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2021. 2, 3
- [6] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021. 3, 7
- [7] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July 10-15, 2018*, pages 1582–1591. PMLR, 2018. 3
- [8] Mathias Gehrig, Willem Aarents, Daniel Gehrig, and Davide Scaramuzza. Dsec: A stereo event camera dataset for driving scenarios. *IEEE Robotics and Automation Letters*, 6(3): 4947–4954, 2021. 1
- [9] Mathias Gehrig, Willem Aarents, Daniel Gehrig, and Davide Scaramuzza. DSEC: A stereo event camera dataset for driving scenarios. *IEEE Robotics and Automation Letters*, 6(3): 4947–4954, 2021. 3
- [10] Thomas Gossard, Julian Krismer, Andreas Ziegler, Jonas Tebbe, and Andreas Zell. Table tennis ball spin estimation with an event camera. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, page 3347–3356. IEEE, 2024. 2, 3
- [11] Rui Graça, Brian McReynolds, and Tobi Delbruck. Shining light on the dvs pixel: A tutorial and discussion about biasing and optimization. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, page 4045–4053. IEEE, 2023. 2, 3
- [12] Michael Grupp. evo: Python package for the evaluation of odometry and slam. <https://github.com/MichaelGrupp/evo>, 2017.
- [13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016. 6, 7
- [14] Javier Hidalgo-Carrio, Guillermo Gallego, and Davide Scaramuzza. Event-aided direct sparse odometry. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2022. 1
- [15] Yuhuang Hu, Shih-Chii Liu, and Tobi Delbruck. v2e: From video frames to realistic dvs events. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1312–1321, 2021. 4
- [16] Damien Joubert, Alexandre Marcireau, Nic Ralph, Andrew Jolley, André Van Schaik, and Gregory Cohen. Event camera simulator improvements via characterized parameters. *Frontiers in Neuroscience*, 15:702765, 2021. 4
- [17] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Židek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. 3
- [18] Simon Klenk, Marvin Motzet, Lukas Koestler, and Daniel Cremers. Deep event visual odometry. In *2024 International Conference on 3D Vision (3DV)*, page 739–749. IEEE, 2024. 5, 6
- [19] Adam Daniel Laud. *Theory and application of reward shaping in reinforcement learning*. PhD thesis, USA, 2004. 3
- [20] Kyunghyun Lee, Ukcheol Shin, and Byeong-Uk Lee. Learning to control camera exposure via reinforcement learning. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 2975–2983. IEEE, 2024. 3, 7
- [21] Gokul B. Nair, Michael Milford, and Tobias Fischer. Enhancing visual place recognition via fast and slow adaptive biasing in event cameras. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 3356–3363. IEEE, 2024. 2, 3
- [22] Etienne Perot, Pierre de Tournemire, Davide Nitti, Jonathan Masci, and Amos Sironi. Learning to detect objects with a 1 megapixel event camera. In *Advances in Neural Information Processing Systems*, pages 16639–16652. Curran Associates, Inc., 2020. 1
- [23] Henri Rebecq, Daniel Gehrig, and Davide Scaramuzza. ESIM: an open event camera simulator. *Conf. on Robotics Learning (CoRL)*, 2018. 4
- [24] Rui Graca, Brian McReynolds, and Tobi Delbruck. Optimal biasing and physical limits of dvs event noise. 2023. 2
- [25] Mehdi Sefidgar Dilmaghani, Waseem Shariff, Cian Ryan, Joe Lemley, and Peter Corcoran. Control and evaluation

- of event cameras output sharpness via bias. In *Fifteenth International Conference on Machine Vision (ICMV 2022)*, page 37. SPIE, 2023. [2](#)
- [26] Takuma Seno and Michita Imai. d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research*, 23(315):1–20, 2022. [7](#)
- [27] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676): 354–359, 2017. [3](#)
- [28] Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. *Robotics: Science and Systems*, 2019. [3](#)
- [29] Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high-speed scenarios. *IEEE Robotics and Automation Letters*, 3(2):994–1001, 2018. [1](#)
- [30] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015. [3](#)
- [31] Yi Zhou, Guillermo Gallego, Xiuyuan Lu, Siqui Liu, and Shaojie Shen. Event-based motion segmentation with spatio-temporal graph cuts. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13, 2021. [1](#)
- [32] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, page 1433–1438. AAAI Press, 2008. [3](#)
- [33] Andreas Ziegler, Daniel Teigland, Jonas Tebbe, Thomas Gossard, and Andreas Zell. Real-time event simulation with frame-based cameras. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023. [4](#)

BiasBench: A reproducible benchmark for tuning the biases of event cameras

Supplementary Material

7. Access to the Dataset

The dataset, the presented metrics as well as some additional information is accessible on our project website:

8. Datasets

Here, we provide some additional information about the three setups of our dataset. The spinning dot in Sec. 8.1, the blinking LED board in Sec. 8.2, and the Visual Odometry (VO) in Sec. 8.3.

8.1. Spinning Dot

The setup of the spinning dot is shown in Fig. 11. A selec-

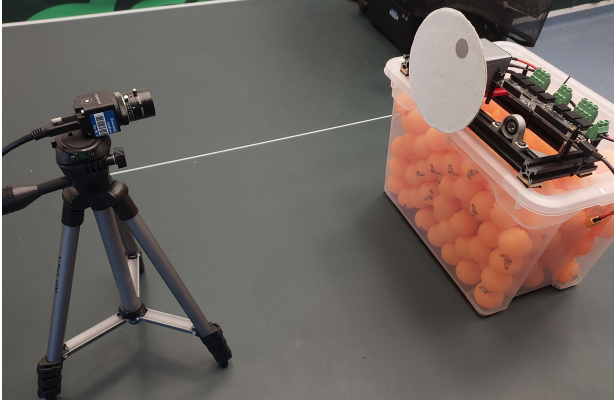


Figure 11. The setup of how we recorded the spinning dot consists of an EVKv4 event-based camera from Prophesee with a resolution of 1280×720 pixels and a white disk with a grey dot connected to a motor. The distance between the motor and the spinning disk is 40 centimeters.

tion of screen captures from the recordings of the grey dot with their bias settings is visualized in Fig. 12. And for the black dot with their bias settings in Fig. 13.

8.2. Blinking LED Board

A blinking LED board, shown in Fig. 14, was specifically designed as an evaluation tool for event cameras in a scientific study. Its purpose is to enable the simultaneous observation of varying frequencies and luminosity gradients, offering a robust testing environment for evaluating event camera performance under diverse conditions.

The board consists of a total of 16 LEDs, categorized as follows:

- **High-Frequency Blinking LEDs (4 LEDs):** These LEDs blink at high frequencies that are not perceivable

by the human eye (>50 Hz). They are particularly sensitive to the *bias_hpf* bias setting, as higher cutoff frequencies in the high-pass filter will predominantly retain events from these high-frequency LEDs while filtering out lower-frequency signals.

- **Sinusoidal Luminosity LEDs (6 LEDs):** These LEDs modulate their brightness following a sinusoidal wave at different frequencies. The smooth, continuous change in luminosity creates a distinct time gradient in the events generated by the camera.
- **Triangular Wave Luminosity LEDs (6 LEDs):** These LEDs also change their brightness at the same set of frequencies as the sinusoidal LEDs, but with a triangular wave pattern. The sharper transitions of the triangular wave offer a different type of temporal gradient for evaluation.

The difference in frequencies across the sinusoidal and triangular LEDs results in varying temporal gradients, allowing for a nuanced assessment of how the event camera responds to different rates of change in light intensity.

The board’s design also allows for testing bias tuning algorithms:

- Adjusting the *bias_diff_on* and *bias_diff_off* parameters can suppress the events generated by low-frequency LEDs, effectively isolating high-frequency activity.
- The *bias_fo* setting provides additional filtering by attenuating signals from lower frequency LEDs, enabling more precise control over the frequency response of the camera.

The LED board is controlled by a microcontroller, allowing easy adjustment of LED frequencies. While Pulse Width Modulation (PWM) is commonly used for LED intensity control, the event camera would detect the rapid on-off transitions. To avoid this, capacitors were added in parallel to the sinusoidal and triangular LEDs, acting as integrators to smooth the PWM signal into a continuous analog waveform. This ensures gradual brightness changes and prevents unwanted high-frequency noise.

We recorded 30,976 sequences, covering the following biases:

- *bias_diff_on*: -80, -60, -40, -20, 0, 20, 40, 60, 80, 100, 120
- *bias_diff_off*: -30, -10, 10, 30, 50, 70, 90, 110, 130, 150, 170
- *refr_time*: 4, 20, 36, 52, 68, 84, 100, 116
- *refr_fo*: -29, -18, -7, 4, 15, 26, 37, 48
- *refr_hpf*: -15, 65, 145, 225

Figure 15 shows the average ER over the whole dataset for each bias setting. The bias combinations with the lowest value for one of the two thresholds contain around 50% of

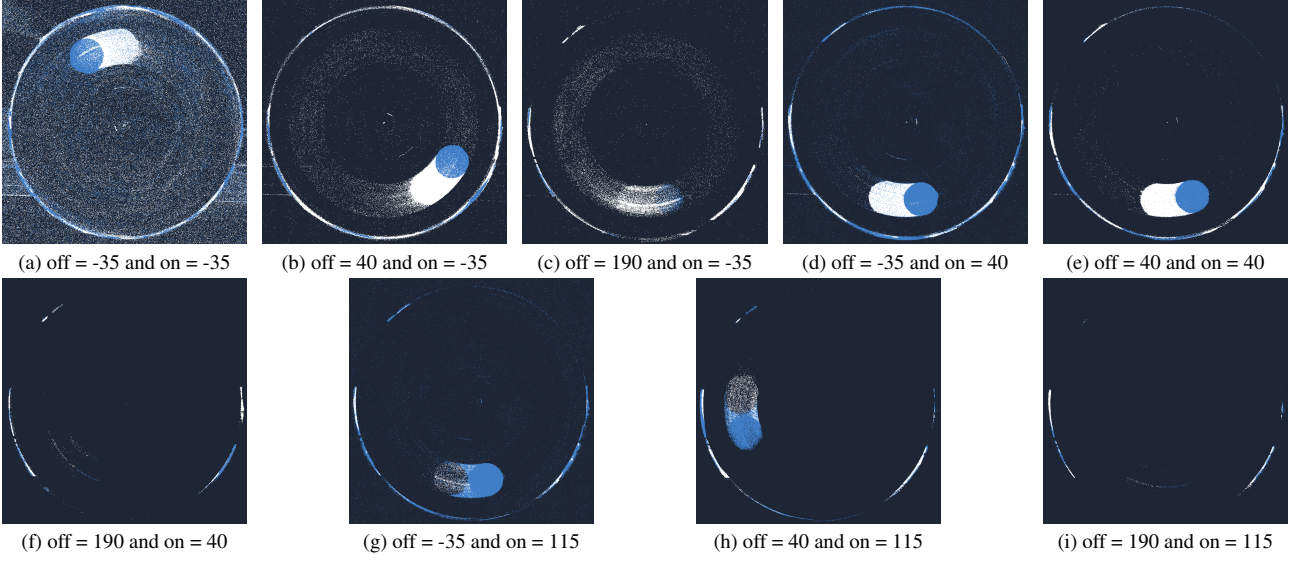


Figure 12. A selection of screen captures of Metavision Studio from 100 recordings of a **grey dot** with their bias settings, where off refers to *bias_diff_off* and on refers to *bias_diff_on*.

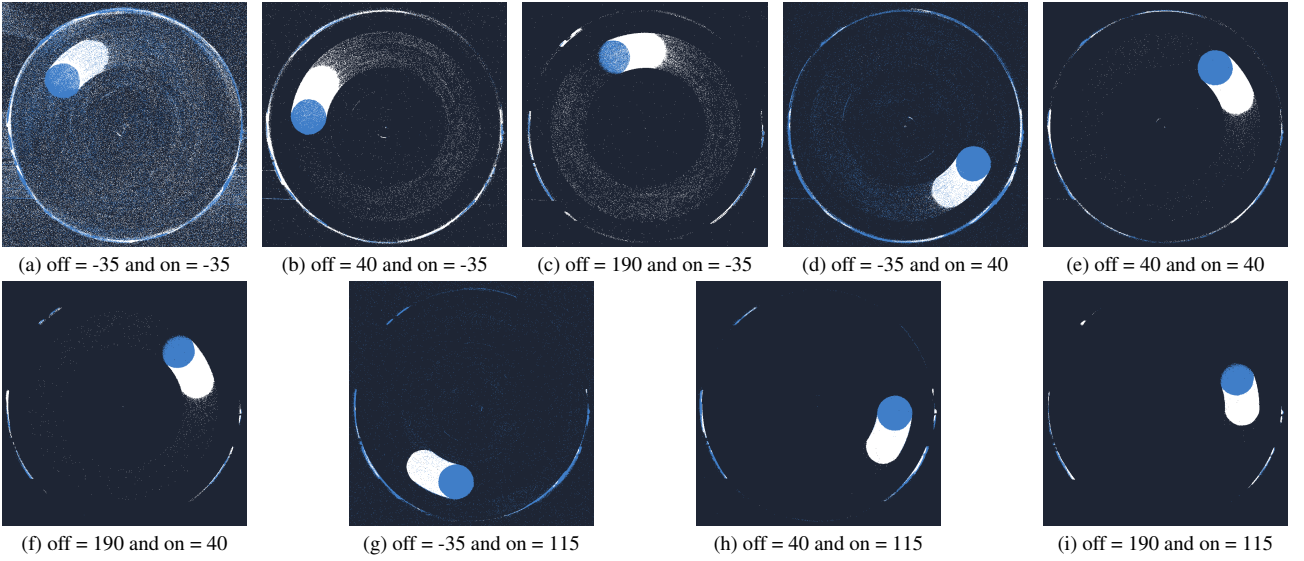


Figure 13. A selection of screen captures of Metavision Studio from 100 recordings of a **black dot** with their bias settings, where off refers to *bias_diff_off* and on refers to *bias_diff_on*.

the total event count. Additionally, a very high high-pass filter can be used to get rid of many events. A decrease in the refractory period also decreases the average event rate as displayed in Figure 16.

The slow-blinking LEDs start to vanish for high bias thresholds. Resulting in an increased RFU in these regions. This can be seen in Fig. 17. The increase of noise in low bias setting also hinders the frequency detection, therefore, an optimal around *bias_diff_off* = -20, *bias_diff_on* = 20

results in the highest average RFU. A strong high-pass filter in combination with a strong low-pass filter also decreases the sensitivity for the detection of some LEDs, therefore, the RFU also increases in these regions significantly.

8.3. Visual Odometry (VO)

To get a repeatable motion and an accurate ground truth trajectory, we mounted an event camera to the end-effector of a Pandal robot arm, shown in Fig. 18, to move the event

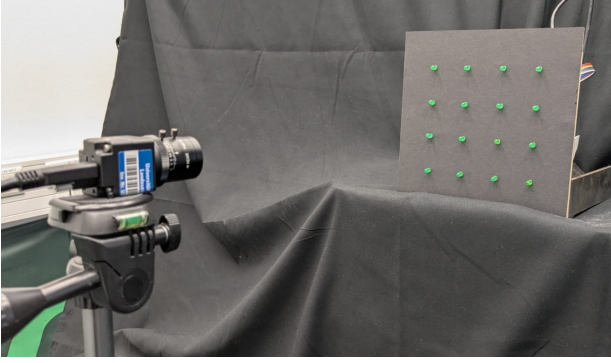


Figure 14. A blinking LED board consisting of LEDs with different frequencies and waveforms.

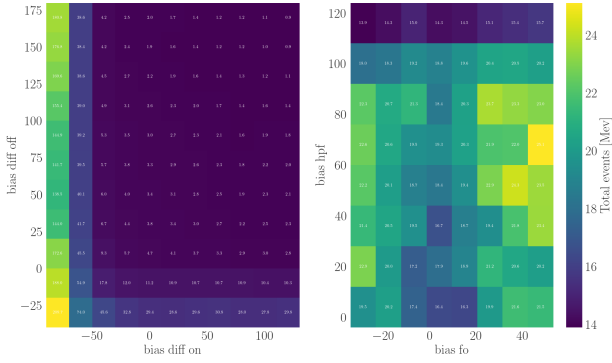


Figure 15. The average event Rate for different bias combinations. *bias_diff_off* vs *bias_diff_on* on the left side, *bias_hpf* vs *bias_fo* on the right side. The lowest threshold bins contain roughly 50% of all events produced.

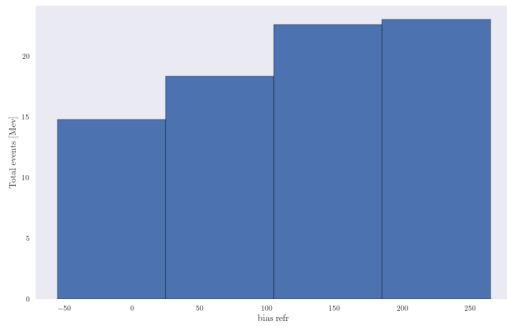


Figure 16. The average event Rate for different refractory periods.

camera along the same trajectory with different biases. The scene was a static environment under constant illumination.

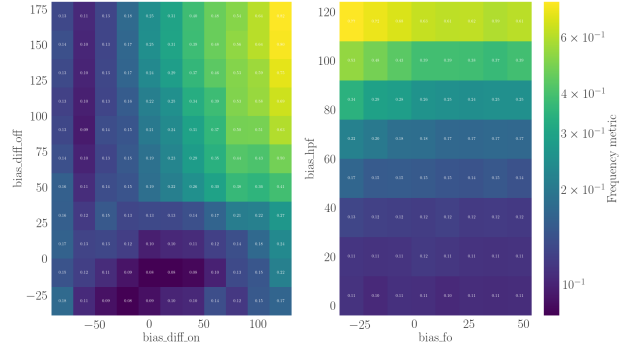


Figure 17. The average RFU for different bias combinations *bias_diff_off* vs *bias_diff_on* on the left side, *bias_hpf* vs *bias_fo* on the right side. The most efficient frequency estimation can be achieved with a *bias_diff_off* = -10, *bias_diff_on* = 20 and *bias_hpf* = 0.

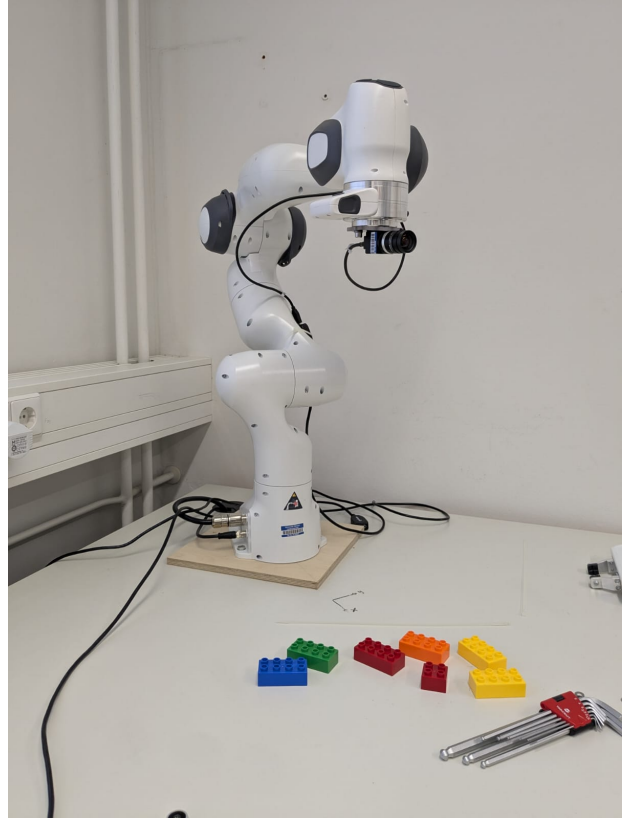


Figure 18. The setup to record the VO dataset.

9. Baseline

The overall pipeline used for this learning process is illustrated in Fig. 19.

In order to rate the output of the event camera, the primary metric we chose was whether the dot on an accumu-

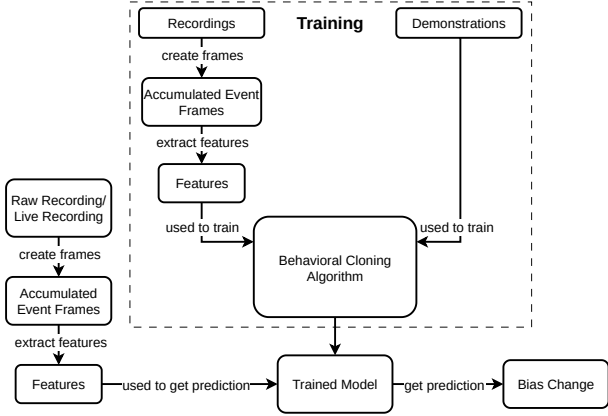


Figure 19. An overview of the Behavioral Cloning (BC) pipeline.

lated event frame is visible to the human eye or not. This is the first and foremost concern because if the dot is not visible, it defeats the entire point of the recording. The secondary metric that we chose is the ER, which should be as low as possible while keeping the signal strong. This translates to low background noise. We define an optimum bias setting as any setting that fulfills the two conditions.

The average ER was calculated for each recording by dividing the total amount of events by the length of the recording. The ER is used to measure the amount of signal and noise and to try to find a bias setting with a good ratio between the two. As can be observed in Fig. 20 and in Fig. 21, for recordings where either of the two biases has a small value, the ER is larger, by order of magnitude, compared to the recordings where the biases had higher values. The recordings with the *bias_diff_on* values between -85 and -60 contain errors because the readout bus of the event camera did not have enough bandwidth to transmit all the events. This means that the ER is bigger than the maximum allowed one.

The most noticeable difference between the heatmap of the ER for the grey dot in Fig. 20 and the one for the black dot in Fig. 21 is that the average ER is higher for the black one. This is expected, as the contrast between the black dot and the white background is higher than for the grey dot.

To find the range of the optimal biases, we also look at the accumulated event frames. An accumulation period of 8 ms has been chosen so that the entire body of the dot can be seen. The first thing noticeable is that for high values of *bias_diff_off*, the dot is no longer visible, as seen in Fig. 12. The camera has such a low sensitivity for those settings that the contrast between the dot and the background is no longer enough to produce any events. As we observed from the ER, for too small values of both biases, especially if they are negative, the noise severely affects the quality of the recording, even if the dot is visible.

For the grey dot, an optimal bias, as we defined it, is *bias_diff_off* = 40 and *bias_diff_on* = 40. For this setting, the dot is visible, and the noise is greatly reduced. We have considered that for the setting *bias_diff_off* = 40 and *bias_diff_on* = 115, the signal was too reduced for it to be considered an optimal bias, even though the dot is visible to the human eye. The observations made from viewing the recordings helped us conclude that optimal biases can be found in the range $[15, 65]$ for *bias_diff_off* and in the range $[40, 90]$ for *bias_diff_on*.

For the black dot, we can observe that the dot is visible even for the maximum values of the thresholds, while the grey one disappears. Because of this, the range of the optimal biases is larger than for the grey dot, being $[40, 165]$ for *bias_diff_off* and $[40, 115]$ for *bias_diff_on*.

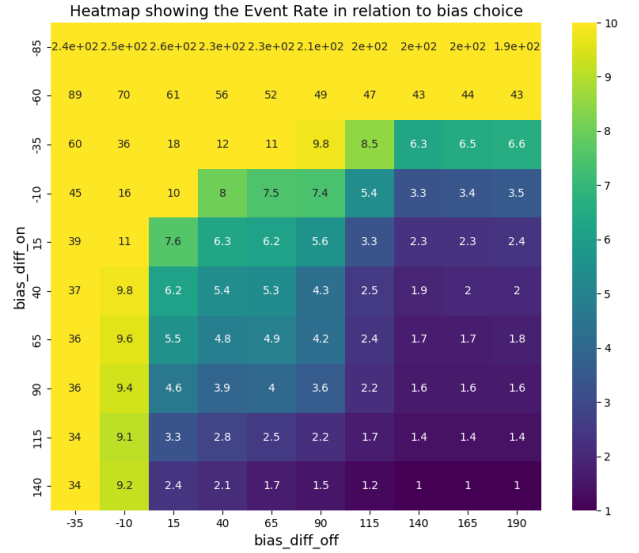


Figure 20. A heatmap showing the Event Rate (ER) for different bias choices of the spinning **grey** dot recording.

We have chosen to define the action in this environment as the relative bias change, expressed with an integer to be added to the current bias value, as this is the way we went about tuning the biases by looking at the output and approximating how big of a change is necessary. Thus, we have written demonstrations consisting of a tuple of actions with the form $(a, b, 0, 0, 0)$, where a represents the change of *bias_diff_off* and b represents the change of *bias_diff_on* that is needed to reach the range of optimal biases.

Different Feature Extractors As an ablation study for the feature extraction, we have adapted two models of ResNet: ResNet18, with 18 layers, and ResNet50, with 50 layers, a more complex network, capable of capturing more information. The choice of the network has an influence on the ability of the network to learn.

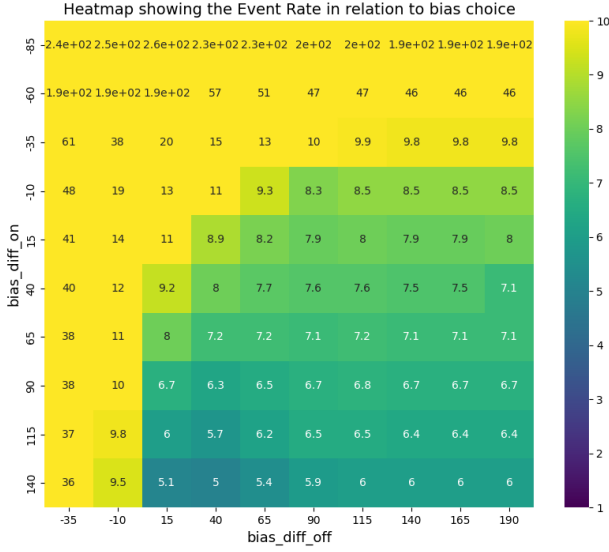


Figure 21. A heatmap showing the Event Rate (ER) for different bias choices of the spinning **black** dot recording.

ResNet50 is a more complex and deep network compared to ResNet18, and therefore, the BC model is able to finish the learning process in 1000 steps. Due to its more complex structure, ResNet50 creates a more information-dense latent space. The downside of this is a more computationally expensive training process for the model, which was already done on ImageNet and is of no concern for this work. The exact loss values after the training process finished are listed in Tab. 6.

As evident from the training and test loss in Tab. 6, ResNet50 is the better option for our purposes, and therefore, we decided to use ResNet50 as our feature extractor.

Model	Test loss
ResNet18	0.24
ResNet50	0.2

Table 6. The final loss values when the training process is finished. The loss is displayed as the Mean Squared Error of the normalized proposed action.

Different Accumulation Times As described in Sec. 4.1, we are using accumulated event frames to extract features from the event stream. In an additional ablation study, we investigated the influence of the accumulation time. Depending on the application, the duration of the accumulation time can have a significant influence on the performance. Therefore, we investigated the influence of different accumulation times (1 ms, 2 ms, 8 ms, and 16 ms).

Accumulation Time	Test loss
1 ms	0.2
2 ms	0.22
8 ms	0.21
16 ms	0.27

Table 7. The final loss values with different accumulation times after the training process finished. The loss is given in the mean squared Error of the normalized proposed action.

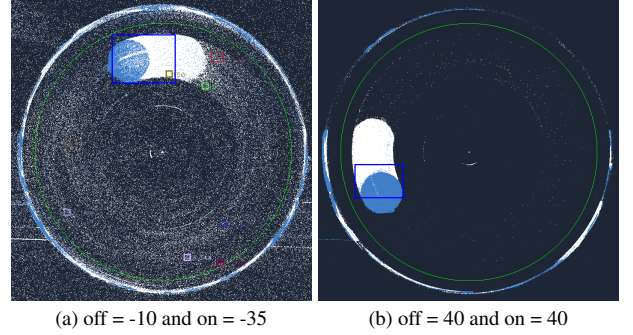


Figure 22. Demonstration of the model proposing a bias change from (a) to (b) that allows the object tracking algorithm to accurately track the dot and not see any other objects caused by noise.

From the final test losses in Tab. 7, we can see that for our data, a shorter accumulation time is favorable.

Appendix D

Detection of Fast-Moving Objects with Neuromorphic Hardware

Authors

Andreas Ziegler , Karl Vetter , Thomas Gossard , Jonas Tebbe , Sebastian Otte , and Andreas Zell

Published in

2025 IEEE International Conference on Robotics and Automation (ICRA 2025)

Time of publication

May 2025

DOI

Included in this dissertation with permission from IEEE.
©2025 IEEE.

Detection of Fast-Moving Objects with Neuromorphic Hardware

Andreas Ziegler¹, Karl Vetter¹, Thomas Gossard¹, Jonas Tebbe¹, Sebastian Otte², and Andreas Zell¹

Abstract—Neuromorphic Computing (NC) and Spiking Neural Networks (SNNs) in particular are often viewed as the next generation of Neural Networks (NNs). NC is a novel bio-inspired paradigm for energy efficient neural computation, often relying on SNNs in which neurons communicate via spikes in a sparse, event-based manner. This communication via spikes can be exploited by neuromorphic hardware implementations very effectively and results in a drastic reductions of power consumption and latency in contrast to regular GPU-based NNs. In recent years, neuromorphic hardware has become more accessible, and the support of learning frameworks has improved. However, available hardware is partially still experimental, and it is not transparent what these solutions are effectively capable of, how they integrate into real-world robotics applications, and how they realistically benefit energy efficiency and latency. In this work, we provide the robotics research community with an overview of what is possible with SNNs on neuromorphic hardware focusing on real-time processing. We introduce a benchmark of three popular neuromorphic hardware devices for the task of event-based object detection. Moreover, we show that an SNN on a neuromorphic hardware is able to run in a challenging table tennis robot setup in real-time.

SUPPLEMENTARY MATERIAL

Additional resources are available at:
<https://cogsys-tuebingen.github.io/snn-edge-benchmark>

I. INTRODUCTION

Spiking Neural Networks (SNNs) mimic the spiking behavior of biological neurons, offering a biologically inspired approach to Neural Network (NN) computation. Unlike traditional artificial neurons that produce real-valued outputs, spiking neurons receive input spikes and integrate them in the state of the neuron, called membrane potential. When the neuron’s membrane potential reaches a defined threshold, the neuron emits a spike that propagates through the network, and resets its membrane potential. More theoretical work on SNNs has already been present in research for decades [1][2][3][4]. With the availability of more computing power through GPUs, it became feasible to train and simulate SNNs for real-world tasks. However, simulating SNNs is very inefficient on GPUs since the Neuromorphic Computing (NC) paradigm is fundamentally different. Therefore, running SNNs on GPUs, though possible, is not a reasonable option for real-world applications, especially not for real-time robotics applications.

¹Andreas Ziegler, Karl Vetter, Thomas Gossard, Jonas Tebbe, and Andreas Zell are with the University of Tübingen. ²Sebastian Otte is with the University of Lübeck. Corresponding author andreas.ziegler@uni-tuebingen.de.

This research was partially funded by Sony AI.

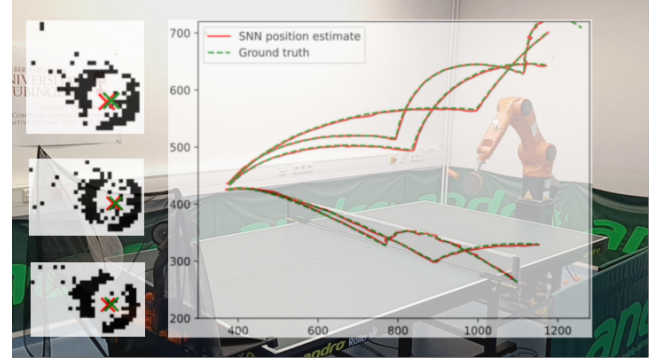


Fig. 1. Left: Three examples of 2D ball detections in an accumulated event frame which serves as the input to the Spiking Neural Network (SNN) with ground truth in green and the estimated position in red. Right: Five observed 2D trajectories in the camera frame of the event-based camera with ground truth in green and the estimated positions in red. Background: The table tennis robot setup with the robot hitting back a table tennis ball in a rally.

In contrast, neuromorphic hardware, specifically designed for efficient SNN processing, can leverage the sparsity and binary nature of SNN outputs, resulting in a drastic reduction of power consumption and latency compared to conventional NNs.

But the efficient processing of SNNs relies on gathering suitable data. Conventional cameras capture frames at a fixed frame rate, providing information about brightness and color. Event-based cameras, on the other hand, report asynchronous brightness changes per pixel, without measuring the absolute brightness [5][6]. These cameras offer a high dynamic range, a temporal resolution in the order of μs , as well as energy and data efficiency. The binary output of event-based cameras aligns with the spike format of SNNs, making them a good match.

This synergy between event-based cameras and SNNs presents an opportunity to improve real-time performance in robotics. One such area where these advancements can be applied is table tennis robotics, which has gained popularity in recent years [7]. While not yet able to compete with professional players, table tennis robots are an exciting research environment to bring algorithms towards their limits. We thus use a table tennis robot scenario as a benchmark suite for event-based neuromorphic perception systems.

A primary perception task for a table tennis robot system is fast and accurate ball detection. So far, most research uses frame-based cameras together with a Convolutional Neural Network (CNN) based ball detection or a classical computer vision approach [8][9][10][11]. While ball detection solutions with frame-based cameras are successfully used, the

high temporal resolution of event-based cameras promises faster and more frequent ball detections. This can improve the prediction of the ball's trajectory, which allows a faster robot control. As mentioned, SNNs align well with event data and can handle more complex scenarios, like a cluttered environment, compared to model-based solutions.

This work explores the combination of an event-based camera and SNNs for object detection and their deployment on neuromorphic edge devices. We analyze the potential benefits of this fusion, and discuss the limitations of a deployment on neuromorphic edge devices. We report the error and run-time, both in simulation and on multiple neuromorphic edge devices, namely the DynapCNN¹ from SynSense, Akida² from BrainChip and Loihi³ from Intel. Benchmarking these three widely used neuromorphic edge devices on our use case of object detection gives an example of what is possible with these edge devices in robotics perception applications.

In summary, our **contributions** with this work are:

- We demonstrate the effectiveness of SNNs for event-based object detection
- We conduct a comparative study of available neuromorphic edge devices
- We present a streamlined integration of event-based sensing, neuromorphic processing, and robot arm planning and control in a table tennis robot setup
- We provide a publicly available benchmark dataset for event-based ball detection

II. RELATED WORK

We will start with an introduction to event-based cameras in Section II-A. SNNs will be covered in Section II-B. Different ways to train an SNN are covered in Section II-C and in Section II-D. We will finish the related work covering spiking object detection in Section II-E.

A. Event-Based Cameras

Event-based cameras, also known as neuromorphic cameras or dynamic vision sensors, have gained considerable attention in computer vision and robotics due to their unique characteristics and advantages over traditional frame-based cameras [5][6]. Event-based cameras operate by detecting logarithmic changes in brightness asynchronously on a per-pixel basis, reporting events only when the brightness change exceeds a specified threshold.

The asynchronous event-based nature of event-based cameras enables high temporal resolution and low latency, making them particularly suitable for capturing fast-moving objects [12][13][14] and scenes with a high dynamic range [15][16].

¹<https://www.synsense.ai/products/dynap-cnn/>

²<https://brainchip.com/akida-enablement-platforms/>

³<https://www.intel.com/content/www/us/en/research/neuromorphic-computing-loihi-2-technology-brief.html>

B. Spiking Neural Networks (SNN)

SNNs have garnered substantial interest in the field of NNs due to their biological inspiration and potential for energy-efficient computation [17]. SNNs differ from traditional Artificial Neural Networks (ANNs) by modeling the spiking behavior of biological neurons more closely. They rely on sparse and binary spikes for information processing. SNNs can represent different neuron models, such as the Integrate-and-Fire (IF) [18] or the leaky IF [19] model.

Studies indicate that SNNs are more energy-efficient [17] and excel in processing spatio-temporal data due to their use of sparse, binary signals [20]. Consequently, SNNs are ideal for energy-sensitive hardware applications [21][22].

A significant advancement in SNN research is the development of efficient training techniques. Converting pre-trained ANNs into SNNs will be described in Section II-C and direct training of SNNs using spike-based learning rules, in Section II-D.

C. Artificial Neural Network (ANN) to SNN Conversion

One approach to train SNNs involves converting pre-trained ANNs into SNNs [23]. This can be done by approximating the output of the Rectified Linear Unit (ReLU) non-linearity with rate-code [24]. Rate-code represents the relative frequency of spikes, obtained by replacing the ReLU with the Heaviside step function and setting a spiking threshold of one. This approximation captures the essential characteristics of the ReLU, for output activities between zero and one. ReLU outputs larger than one can not be represented by normal IF neurons, as they can at most spike once per time step. However, a multi-spike neuron model [25] allows the direct representation of the entire ReLU output space through rate-code.

D. Direct Spiking Neural Network (SNN) Training

There are two major lines of research to directly train SNNs. One is to train the model through biologically plausible local learning rules, e.g., Spike Timing Dependent Plasticity (STDP) [26][27], the other one to use so-called surrogate gradients [28]. Since STDP is not available for most of the SNN frameworks used in this work, we will focus on surrogate gradients.

Back-propagation in SNNs is akin to a specific instance of back-propagation through time (BPTT), typical for RNN training. However, the Heaviside function lacks differentiability at the threshold, and therefore, surrogate gradients are necessary for approximating gradients [28].

E. Spiking Object Detection

Object detection is a fundamental task in computer vision, and traditional approaches typically use CNNs. In recent years, there has been growing interest in exploring SNNs for object detection tasks due to their potential for improved energy efficiency and event-driven processing [21][22].

The YOLO architecture [29], a popular object detection framework, has been successfully converted into a SNN variant called Spiking-YOLO in [30].

SNNs were also explored for specific object detection tasks. In [31], SNNs were employed to predict ball trajectories using data from event-based cameras. Their approach utilized spatio-temporal filters with weights incorporating delays to capture temporal information. They also employed leaky IF neurons and trained the network using STDP.

An SNN was utilized in [32] for detecting balls and pipes in a simulated environment. They used simulated neuromorphic data and designed an SNN architecture specifically tailored for detecting circles, ellipses, and lines using the Hough transform. Their approach demonstrated successful detection using the designed SNN without training, relying on hard-coded algorithms.

Although promising, SNNs for object detection still faces challenges, including higher error rates compared to CNN-based detectors. Further advancements in network architectures, training algorithms, spike-based encoding schemes, and hardware are necessary to bridge this performance gap. Moreover, optimizing the computational and memory efficiency of SNN models is crucial for enabling real-time object detection on resource-constrained devices.

III. METHOD

We recorded a ball detection dataset to train our SNNs, described in Section III-A. In this work, we used three state-of-the-art SNN frameworks and a corresponding neuromorphic edge device. For each of them, we designed an SNN architecture, conforming with their specific constraints, described in Section III-B. The training with the three SNN frameworks is covered in Section III-C.

A. Data

We describe the recording setup in Section III-A.1, followed by an explanation of how the training labels were generated in Section III-A.2. The specifications of the dataset are summarized in Table I.

1) *Recording setup*: We used a table tennis robot setup as introduced in [8]. The setup was extended with two event-based cameras as described in [7]. This camera system consists of four FLIR Chameleon3 frame-based cameras (140 fps, 1280x1024 pixels) and two Prophesee EVK4 event-based cameras (1280x720 pixels). Camera bias settings for the event-based camera were configured to minimize noise, and so that most events would be caused by the flying ball. The whole table tennis robot system is visualized in Fig. 2. To calibrate this camera system containing frame- and event-based cameras, we used the wand-based calibration approach introduced in [33]. The frame-based cameras are used to detect and triangulate the position of the table tennis ball in

3D, as described in [8]. The event stream from the event-based cameras is used as input for our SNN approach.

A Butterfly Amicus Prime ball gun, with default speed settings (4m/s), was used to shoot balls, as shown in the background of Fig. 1.

2) *Data generation*: The input of our SNNs is a matrix of size 64x64 pixels, where over a time range of 1ms the pixel value is set to one if at least one event occurred at the pixel and zero otherwise. To reduce computation and achieve faster inference times, we favored smaller networks and thus did not include the polarity information of the events. As in [34], our event representation does also not differentiate between pixels with more or less events, as long as they have any. This reduction in information was carried out as it led to better performance in our experiments. A dynamic Region of Interest (ROI) was used to crop 64x64 pixels out of the 1280x720 pixels output of the event-based camera, similar to [8]. This was done by setting the last known ball position as the center of the ROI. This has the benefit of being applicable in real-time using the system's own past ball position estimates. Since the system starts each trajectory without past position estimates, we fixed the start region. This limitation could be alleviated by using the position of a frame-based detection system or an CNN based ball detector for the initialization.

The output of the SNNs is the 2D pixel position of the detected ball. Therefore, the required ground truth is the 2D ball position in the image frame of the event-based camera. We used two ways to generate the 2D ball positions, serving as ground truth. We projected the 3D ball positions from the frame-based camera system into the camera frame of the event-based cameras. This way, we generated 7569 ground truth positions. Since we only obtain 3D positions every 7ms, we additionally labeled the 2D ball position manually on an additional set of 2123 event frames.

B. Network Architecture

The state-of-the-art SNN frameworks and their corresponding neuromorphic edge device, we used in this work,

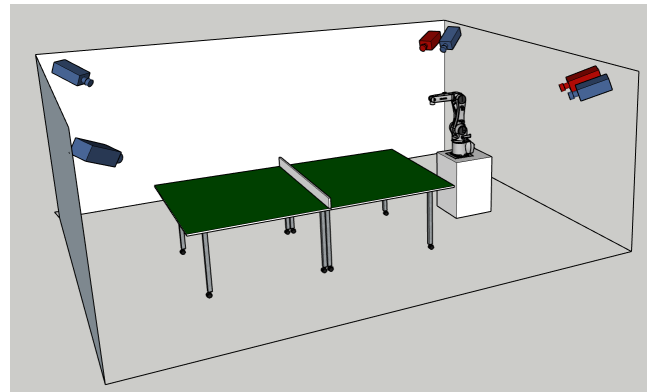


Fig. 2. Our camera setup consisting of four frame-based cameras (in blue) and two event-based cameras (in red) with baselines of 3m to 5m. Schematic is up to scale.

TABLE I
DATASET SPECIFICATIONS

Number of samples	Training: 8630, validation: 531, test: 531
Labeled samples	Automatic labeled: 7569, manually labeled: 2123
Cameras used	Event-based: 2x Prophesee EVK4 (1280x720) Frame-based: 4x FLIR Chameleon3 (1280x1024, 140fps)
Event accumulation time	1ms
Ballgun	Butterfly Amicus Prime, ball speed 4m/s

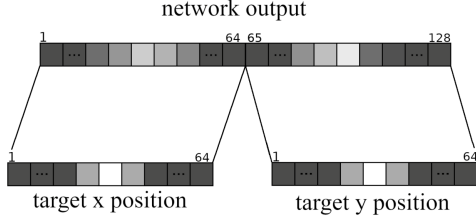


Fig. 3. The network has 128 output neurons, which are split into two populations. Each neuron in the first population represents an x -position, and each neuron in the second represents a y -position. The target is also split into an x - and y -target, each setting one as the target activity for the correct neuron, 0.5 for the two adjacent neurons, and zero for all others. Values are represented using brightness, with larger values being brighter.

TABLE II

NETWORK ARCHITECTURE FOR THE THREE SNN FRAMEWORKS. THE DETAILS ARE GIVEN IN THE FOLLOWING ORDER: SINABS / META TF / LAVA. “NONE” INDICATES THAT NO LAYER IS USED FOR THE CORRESPONDING SNN FRAMEWORK. WHERE STRIDE IS NOT EXPLICIT, IT DEFAULTS TO 1.

Layer	Layer Specifics
1	ConvLayer(outChannels = 4, kernelsize = 5x5, stride = 2) Multispike IF / (BatchNorm & QuantizedReLU) / LIF AveragePool(2x2) / MaxPool(2x2) / None
2	ConvLayer(outChannels = 4, kernelsize = 3x3) Multispike IF / (BatchNorm & QuantizedReLU) / LIF AveragePool(2x2) / None / None
3	LinearLayer(outChannels = 64) Multispike IF / (BatchNorm & QuantizedReLU) / LIF
4	LinearLayer(outChannels = 128) Multispike IF / None / LIF

each have their constraints. Some constraints are the input/output resolutions per layer, the number of output weights per neuron, and the type of supported layers. These constraints make it difficult to run any more complex object detection network. Therefore, the challenge was to find a trade-off between the network’s complexity and the constraints and limitations of the neuromorphic edge devices.

We cast the task of detecting the table tennis ball in 2D as a classification task. This allows fewer time steps and, therefore, a faster inference time than a regression network due to the rate-code approximation. We treat the x - and y -positions as two independent classification tasks. In this case, the ball’s x -position can be one of 64 classes, as can the y -position. A visualization of the network’s output is shown in Fig. 3. Although the neuromorphic edge devices support higher input resolutions, we decided to use only 64x64 pixels to improve the inference time, which is crucial for our real-time use case. To receive the network’s prediction, the neurons with the maximal output are determined in each of the two populations.

The network consists of four layers, the first two being convolutional layers and the last two being linear layers. A description of the network is provided in Table II. The activation function and the pooling are given in the following order: sinabs / MetaTF / Lava. “None” indicates that no layer is used for the corresponding SNN framework. Since all the

SNN frameworks we used had different limitations, we could not use one SNN architecture for all of them. Therefore, we modified the SNN to comply with the constraints of the corresponding framework.

1) *sinabs (DynapCNN)*: While the SNN trained on the GPU processes frames and is clocked, the SNN on the DynapCNN processes events and is asynchronous. This difference leads to a gap between the two SNNs. We use the multi-spike learning approach [25] supported by sinabs, since it reduces this gap. The multispike IF neurons are followed by average pooling to further reduce network size and remain within the limits for the number of neurons per layer imposed by the DynapCNN.

2) *MetaTF (Akida)*: The network we used to train with the MetaTF framework uses batch normalization layers in all layers except the last one, as it led to improved performance, and Akida supports the use of biases, which are necessary to integrate batch normalization into an SNN without significant effect on the inference time. Quantized ReLUs are used because the Akida chip squashes the rate-code approximation of the ReLU into one time step, where it is then represented by a step-wise quantized ReLU. In the first layer, we use max pooling instead of average pooling, since this is the pooling type supported by the MetaTF framework. For the use on the chip, pooling comes with additional restrictions, which is why there is no second pooling layer and instead a stride of two in the second convolutional layer.

3) *Lava (Loihi2)*: For the network trained with Lava, we use leaky IF neurons, since they are the standard neuron type supported by the Loihi2 chip and performed better than IF neurons. An increased number of channels (8, 16, 64, and 128) is used, as this was necessary to reach comparable accuracy. No pooling layers were used, as they cannot be mapped onto the Loihi2 chip.

C. Training

As mentioned in Section II-B, there are two ways to train SNNs. The conversion from an ANN to an SNN and direct SNN training. The former approach has notable drawbacks, including a typical decline in accuracy, especially when using few time steps, and a high spike rate [35]. Therefore, we used direct SNN training for the DynapCNN and Intel’s bootstrap method for Loihi2. Akida uses an equivalent representation of an SNN using a step-wise quantized ReLU, which can be trained using quantization-aware training and allows for processing in a single time step.

While the standard loss for a classification task would be the cross entropy loss, the Mean Squared Error (MSE) proved more suitable for our use case. We observed that the cross entropy loss leads to SNNs with very large activations. Large activations result in more spikes, causing more synaptic operations, making the network slower when used on edge devices. Using the MSE requires a target output of the same shape as the network output. The target that worked best was setting the correct neuron’s target output to one, the target output for the two neurons directly adjacent to 0.5, and all other neuron targets to 0, as visualized in Fig. 3.

TABLE III
COMMON HYPERPARAMETERS OF THE SNN FRAMEWORKS.

SNN framework	Learning rate	Batch size	Optimizer
sinabs (DynapCNN)	0.0001	200	Adam
Lava (Loihi2)	0.001	100	Adam
MetaTF (Akida)	0.0001	1000	Adam

For all SNN frameworks, we trained the networks on a GPU and deployed them on the edge devices for inference. We used all the 7569 automatically generated 2D ground truth positions and additional 1061 manually labeled positions (as described in Section III-A.2) for the training set and 531 manually labeled 2D positions for the validation and the test set. Since the training depends on the SNN framework, we describe the training for each one. Common hyperparameters are listed in Table III.

1) *Training with sinabs (DynapCNN)*: To train SNNs for the DynapCNN, SynSense provides sinabs⁴ as a framework. We use the multi-spike learning approach [25] supported by sinabs, since it closely matches the neural behavior on the chip. The sinabs framework additionally allows for the tracking of the network’s synaptic operations, which can then be used as an additional loss term to prevent over activated neurons on the DynapCNN. With the multi-spike approach, the number of generated spikes when the membrane potential exceeds the threshold is proportional to the membrane potential.

To minimize the quantization loss, the distribution of weights needs to be such that no extreme outlier weights exist, which would decrease the quantization accuracy for all other weights. To normalize the weights on the DynapCNN, we used an additional loss term using only the absolute maximum of the weights for every layer. The SNN was trained using the periodic exponential function as surrogate gradient function [28] and backprop through time.

2) *Training with MetaTF (Akida)*: To train SNNs for the Akida, BrainChip provides MetaTF⁵ as a framework. MetaTF directly trains an ANN approximating a SNN through quantization. We followed the guidelines of the MetaTF documentation to train the SNN with MetaTF.

3) *Training with Lava (Loihi2)*: To train SNNs for the Loihi2, Intel provides Lava⁶ as a framework. Direct training is supported by Lava with an enhanced version of the SLAYER framework [36]. We followed the guidelines of the Lava documentation to train the SNN with Lava-dl. The neuron thresholds were set to 0.25 and voltage decay to 0.05.

IV. EXPERIMENTS

In the first experiment, described in Section IV-A, we evaluate the detection error and measure the time per forward pass of the three different SNN frameworks in simulation as well as on the neuromorphic edge device. Section IV-B describes the second experiment, in which we run the SNN

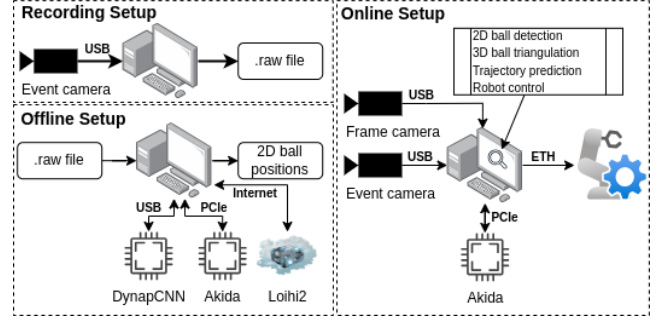


Fig. 4. Left: Experiment setup for the offline experiment. Right: Experiment setup for the online experiment.

TABLE IV
BALL CENTER ERROR AND TIME PER FORWARD PASS OF THE DIFFERENT SNN FRAMEWORKS AND ON THEIR CORRESPONDING NEUROMORPHIC EDGE DEVICE. STEPS INDICATE HOW MANY TIME STEPS WERE USED FOR ONE INFERENCE PASS. THE RESULTS ARE REPORTED WITH MEAN AND STD. DEV OVER MULTIPLE RUNS.

Framework	Sim	Hardware	Steps	Error [pixel]	One forward pass [ms]
BrainChip	✓		15 (1)	1.44 ± 1.41	1.75 ± 0.82
sinabs	✓		8 (1)	1.55 ± 1.89	1.59 ± 1.61
Lava	✓		20	1.57 ± 1.55	1764.89 ± 1374.36
BrainChip		Akida	15 (1)	1.44 ± 1.41	2.20 ± 0.35
sinabs		DynapCNN	8	1.59 ± 1.83	46.04 ± 21.38
Lava		Loihi2	20	1.57 ± 1.55	1458.57 ± 230.500
TensorFlow		RTX 2080 Ti	-	3.55 ± 1.49	3.49 ± 0.62

on a neuromorphic edge device within a table tennis robot setup in real-time.

A. Offline experiment

In this experiment, we compare the different SNN frameworks in terms of error and time per forward pass, given our robotic perception task. Using our recorded data, we ran the 2D ball detection 10 times for BrainChip and sinabs and 5 times for Lava. The setup is depicted in Fig. 4 on the left. Five recorded 2D ball trajectories and three 2D ball detections are shown in Fig. 1.

The error, as well as the time per forward pass, was recorded for every run, and we report the mean and standard deviation in Table IV. Steps indicate how many time steps were used for the rate-code for one inference pass, a trade-off between speed and accuracy. The steps count in brackets represents the number of time steps when using multispikes neurons. We also report the results of an existing setup [37], based on [10], as a comparison. This setup uses a frame-based camera and a CNN with similar image resolution and number of convolutional layers on an RTX 2080 Ti GPU.

As can be seen in Table IV the BrainChip framework has the lowest ball center error of 1.44 pixels. The error of sinabs and Lava are 1.55 pixels and 1.57 pixels, respectively, quite close together.

Important to note is that depending on the neuromorphic edge device, the total time per forward pass is heavily influenced by the integration into the system. The total time for a forward pass includes the inference time and, depending on the neuromorphic edge device data transfer, pre- and post-

⁴<https://github.com/synsense/sinabs>

⁵<https://doc.brainchipinc.com/index.html>

⁶<https://lava-nc.org/index.html>

TABLE V

TOTAL TIME PER FORWARD PASS AND INFERENCE TIME OF THE DIFFERENT NEUROMORPHIC EDGE DEVICES WITH MEAN AND STD. DEV OVER 10 RUNS AND THE POWER CONSUMPTION. [†]PROCESSING ONE INPUT ON THE DYNAPCNN TAKES AN AVERAGE OF 1.765MS, BUT WITH DELAYS FROM THE USB CONNECTION, THE TIME FOR A FORWARD PASS IS SIGNIFICANTLY LONGER. *TAKEN FROM SPECS.

Device	One forward pass [ms]	Inference time [ms]	Power consumption [mW]
Akida	2.20 ± 0.35	0.89 ± 0.28	~ 4.5
DynapCNN	46.04 ± 1.38 [†]	0.82 ± 0.24	~ 5*
Loihi2	1458.57 ± 230.50	0.62 ± 0.01	~ 100*
RTX 2080 Ti	3.49 ± 0.62	1.77 ± 0.28	~ 250000*

processing. Therefore, we list the total time per forward pass, the inference time, and the power consumption in Table V.

For the BrainChip Akida, which is installed as a PCIe card, the time to load data onto the device and return the results is a relatively small overhead (0.89ms inference time and 2.20ms for a forward pass).

The DynapCNN from SynSense is connected to the PC via USB. We operate the DynapCNN in a streaming mode. This allows a fast inference time of 0.82ms but does come with a relatively high delay coming from the USB connection and other sources, which leads to 46.04ms for a forward pass. The streaming mode allows for a high data throughput with an average processing time of 1.765ms on the chip but does not remove the delay caused by the USB connection. Using the DynapCNN with another event-based camera might improve the setup and therefore, remove the mentioned delays. The recent development of combining an event-based camera with an SNN on the same chip, as the Speck⁷ from SynSense, looks promising. However, the current camera resolution of 128x128 pixels is too low for our application.

For Loihi2, we did not have direct access to an edge device, but only through a virtual machine provided by Intel via the Intel Neuromorphic Research Community⁸. With this Loihi2 setup, the time per forward pass is with 1458.57ms two orders of magnitude slower than the other devices. This is mainly due to inefficient communication between the virtual machine and the Loihi2 chip, which is a consequence of our current implementation not being well optimized in this regard. Note that the inference time on the GPU is longer than on all neuromorphic edge devices, however, the good hardware integration of the GPU shows its advantage.

Regarding the power consumption, we measured the one for Akida, but had to rely on specs for the other devices. The power consumption for Akida and DynapCNN is quite close together. Loihi2, being a much more powerful and versatile platform, consumes an order of magnitude more energy, still several orders of magnitude lower than a GPU, showing the benefits of Neuromorphic Computing (NC) on neuromorphic edge devices for robotic applications.

Overall, we can see that while the errors of the different neuromorphic frameworks are similar, the difference in infer-

ence time and the time per forward pass varies significantly. As already mentioned, the way in which the neuromorphic edge devices are connected to the PC plays a crucial role.

B. Online (real-time) experiment

In this experiment, we integrated the SNN based ball detection into our table tennis robot system, described in Section III-A. As explained in [8], to determine the 3D position of the table tennis ball, at least two cameras need to detect the ball in order to perform triangulation. Since we only have one device for each of the three SNN frameworks, covered in this work, we used one event-based camera with the introduced SNN ball detection together with a frame-based camera and the ball detection presented in [8]. We used the Akida PCIe card from BrainChip, since the system integration does not introduce latencies as high as the DynapCNN does, and we did not yet have access to a Loihi2 edge device. The experimental setup is depicted in Fig. 4 on the right, and the physical setup is shown in the background of Fig. 1. In this experiment, we shot 15 table tennis balls with a Butterfly Amicus ball gun. Since the 3D triangulation and robot arm control are not part of this work, we used the existing ones from [8]. We report the ball return rate of the robot, since controlling the landing point of the ball is out of scope for this work. Given this setup, we achieved a ball return rate of 1.0.

As previously mentioned, we used one event-based camera with the SNN based ball detection and a frame-based camera with an existing ball detection to triangulate the 3D ball position. Although our online experiment does not rely solely on our SNN based ball detection, we have shown that this would be possible, and future work could involve evaluating the system with an additional Akida PCIe card.

V. CONCLUSION

With neuromorphic hardware more accessible in recent years, Neuromorphic Computing (NC) and Spiking Neural Networks (SNNs) in particular has become more relevant for robotics. In this work, we used an event camera in combination with SNNs for ball detection as a real-time perception task. Three different SNN frameworks, namely sinabs, MetaTF, and Lava, and a corresponding neuromorphic edge device (DynapCNN, Akida, and Loihi2) were used, and their errors, time per forward pass, inference time, and power consumption were compared. Moreover, we show that an SNN on a neuromorphic hardware is able to run in a challenging table tennis robot setup in real-time.

Despite the promise of asynchronous processing with SNNs, the current edge devices face limitations, primarily attributed to hardware integration. Our results show that the better a neuromorphic edge device is connected to the main compute unit, e.g., as a PCIe card, the better the overall run-time.

This work aims to provide the robotic research community with insights into the possibilities and challenges of deploying SNNs on current neuromorphic edge devices for real-time robotic applications.

⁷<https://www.synsense.ai/products/speck-2/>

⁸<https://intel-ncl.atlassian.net/wiki/spaces/INRC/overview>

REFERENCES

- [1] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.
- [2] H. Paugam-Moisy and S. M. Bohté, "Computing with spiking neuron networks," *Handbook of natural computing*, vol. 1, pp. 1–47, 2012.
- [3] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," *Advances in neural information processing systems*, vol. 31, 2018.
- [4] S. Higuchi, S. Kairat, S. Bohte, and S. Otte, "Balanced resonate-and-fire neurons," in *Forty-first International Conference on Machine Learning*, 2024.
- [5] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120 dB 15 us latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008. [Online]. Available: <https://doi.org/10.1109/jssc.2007.914337>
- [6] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Tabá, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–180, 2022.
- [7] A. Ziegler, T. Gossard, K. Vetter, J. Tebbe, and A. Zell, "A multi-modal table tennis robot system," in *RoboLetics: Workshop on Robot Learning in Athletics @CoRL 2023*, 2023. [Online]. Available: <https://arxiv.org/abs/2310.19062>
- [8] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, "A table tennis robot system using an industrial KUKA robot arm," in *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 33–45. [Online]. Available: https://doi.org/10.1007/978-3-030-12939-2_3
- [9] D. D'Ambrosio, N. Jaitly, V. Sindhwani, K. Oslund, P. Xu, N. Lazic, A. Shankar, T. Ding, J. Abelian, E. Coumans, G. Kouretas, T. Nguyen, J. Boyd, A. Iscen, R. Mahjourian, V. Vanhoucke, A. Bewley, Y. Kuang, M. Ahn, D. Jain, S. Kataoka, O. Cortes, P. Sermanet, C. Lynch, P. Sanketi, K. Choromanski, W. Gao, J. Kangasputa, K. Reymann, G. Vesom, S. Moore, A. Singh, S. Abeyruwan, and L. Graesser, "Robotic table tennis: A case study into a high speed learning system," in *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation, Jul. 2023. [Online]. Available: <https://doi.org/10.15607/rss.2023.xix.006>
- [10] S. Gomez-Gonzalez, Y. Nemmour, B. Schölkopf, and J. Peters, "Reliable real-time ball tracking for robot table tennis," *Robotics*, vol. 8, no. 4, p. 90, Oct. 2019. [Online]. Available: <https://doi.org/10.3390/robotics8040090>
- [11] T. Ding, L. Graesser, S. Abeyruwan, D. B. D'Ambrosio, A. Shankar, P. Sermanet, P. R. Sanketi, and C. Lynch, "Learning high speed precision table tennis on a physical robot," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2022. [Online]. Available: <https://doi.org/10.1109/iros47612.2022.9982205>
- [12] M. Monforte, A. Arriandiaga, A. Glover, and C. Bartolozzi, "Exploiting event cameras for spatio-temporal prediction of fast-changing trajectories," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, Aug. 2020. [Online]. Available: <http://dx.doi.org/10.1109/AICAS48895.2020.9073855>
- [13] A. Mitrokhin, C. Fermüller, C. Parameshwara, and Y. Aloimonos, "Event-based moving object detection and tracking," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2018. [Online]. Available: <http://dx.doi.org/10.1109/IROS.2018.8593805>
- [14] B. Forrai, T. Miki, D. Gehrig, M. Hutter, and D. Scaramuzza, "Event-based agile object catching with a quadrupedal robot," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2023. [Online]. Available: <http://dx.doi.org/10.1109/ICRA48891.2023.10161392>
- [15] E. Perot, P. de Tournemire, D. Nitti, J. Masci, and A. Sironi, "Learning to detect objects with a 1 megapixel event camera," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 16 639–16 652. [Online]. Available: <https://proceedings.neurips.cc/paper/2020/file/c213877427b46fa96cff6c39e837ccee-Paper.pdf>
- [16] T. Stoffregen, C. Scheerlinck, D. Scaramuzza, T. Drummond, N. Barnes, L. Kleeman, and R. Mahony, *Reducing the Sim-to-Real Gap for Event Cameras*. Springer International Publishing, 2020, p. 534–549. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-58583-9_32
- [17] E. Lemaire, L. Cordone, A. Castagnetti, P.-E. Novac, J. Courtois, and B. Miramond, *An Analytical Estimation of Spiking Neural Networks Energy Efficiency*. Springer International Publishing, 2023, p. 574–587. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-30105-6_48
- [18] N. Brunel and M. C. W. van Rossum, "Lapicque's 1907 paper: from frogs to integrate-and-fire," *Biological Cybernetics*, vol. 97, no. 5–6, pp. 337–339, Oct. 2007. [Online]. Available: <https://doi.org/10.1007/s00422-007-0190-0>
- [19] S. Lu and F. Xu, "Linear leaky-integrate-and-fire neuron model based spiking neural networks and

- its mapping relationship to deep neural networks,” *Frontiers in Neuroscience*, vol. 16, Aug. 2022. [Online]. Available: <http://dx.doi.org/10.3389/fnins.2022.857513>
- [20] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, “Benchmarking keyword spotting efficiency on neuromorphic hardware,” in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, ser. NICE ’19. ACM, Mar. 2019. [Online]. Available: <http://dx.doi.org/10.1145/3320288.3320304>
- [21] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, p. 82–99, Jan. 2018. [Online]. Available: <http://dx.doi.org/10.1109/MM.2018.112130359>
- [22] 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,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, p. 1537–1557, Oct. 2015. [Online]. Available: <http://dx.doi.org/10.1109/TCAD.2015.2474396>
- [23] Y. Cao, Y. Chen, and D. Khosla, “Spiking deep convolutional neural networks for energy-efficient object recognition,” *International Journal of Computer Vision*, vol. 113, no. 1, p. 54–66, Nov. 2014. [Online]. Available: <http://dx.doi.org/10.1007/s11263-014-0788-3>
- [24] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing,” in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Jul. 2015. [Online]. Available: <http://dx.doi.org/10.1109/IJCNN.2015.7280696>
- [25] P. Weidel and S. Sheik, “Wavesense: Efficient temporal convolutions with spiking neural networks for keyword spotting,” 2021. [Online]. Available: <https://arxiv.org/abs/2111.01456>
- [26] P. U. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in Computational Neuroscience*, vol. 9, Aug. 2015. [Online]. Available: <http://dx.doi.org/10.3389/fncom.2015.00099>
- [27] G.-q. Bi and M.-m. Poo, “Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type,” *The Journal of Neuroscience*, vol. 18, no. 24, p. 10464–10472, Dec. 1998. [Online]. Available: <http://dx.doi.org/10.1523/JNEUROSCI.18-24-10464.1998>
- [28] E. O. Neftci, H. Mostafa, and F. Zenke, “Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks,” *IEEE Signal Processing Magazine*, vol. 36, no. 6, p. 51–63, Nov. 2019. [Online]. Available: <http://dx.doi.org/10.1109/MSP.2019.2931595>
- [29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [30] S. Kim, S. Park, B. Na, and S. Yoon, “Spiking-YOLO: Spiking neural network for energy-efficient object detection,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, pp. 11 270–11 277, Apr. 2020. [Online]. Available: <https://doi.org/10.1609/aaai.v34i07.6787>
- [31] G. Debat, T. Chauhan, B. R. Cottureau, T. Masquelier, M. Paindavoine, and R. Baures, “Event-based trajectory prediction using spiking neural networks,” *Frontiers in Computational Neuroscience*, vol. 15, May 2021. [Online]. Available: <https://doi.org/10.3389/fncom.2021.658764>
- [32] Z. Jiang, Z. Bing, K. Huang, and A. Knoll, “Retina-based pipe-like object tracking implemented through spiking neural network on a snake robot,” *Frontiers in Neurobotics*, vol. 13, May 2019. [Online]. Available: <https://doi.org/10.3389/fnbot.2019.00029>
- [33] T. Gossard, A. Ziegler, L. Kolmar, J. Tebbe, and A. Zell, “ewand: A calibration framework for wide baseline frame-based and event-based camera systems,” in *2024 International Conference on Robotics and Automation (ICRA)*. IEEE, 2024. [Online]. Available: <https://arxiv.org/pdf/2309.12685.pdf>
- [34] V. Vasco, A. Glover, and C. Bartolozzi, “Fast event-based harris corner detection exploiting the advantages of event-driven cameras,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2016. [Online]. Available: <http://dx.doi.org/10.1109/IROS.2016.7759610>
- [35] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, “Training spiking neural networks using lessons from deep learning,” *Proceedings of the IEEE*, vol. 111, no. 9, p. 1016–1054, Sep. 2023. [Online]. Available: <http://dx.doi.org/10.1109/JPROC.2023.3308088>
- [36] S. B. Shrestha and G. Orchard, “Slayer: Spike layer error reassignment in time,” in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31.

Curran Associates, Inc., 2018. [Online]. Available:
[https://proceedings.neurips.cc/paper_files/paper/2018/
file/82f2b308c3b01637c607ce05f52a2fed-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2018/file/82f2b308c3b01637c607ce05f52a2fed-Paper.pdf)

- [37] J. Tebbe, "Adaptive robot systems in highly dynamic environments: A table tennis robot," Ph.D. dissertation, University of Tübingen, 2021.

Appendix E

Table tennis ball spin estimation with an event camera

Authors

Thomas Gossard*, Julian Krismer*, Andreas Ziegler, Jonas Tebbe
and Andreas Zell

Published in

2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW 2024)

Time of publication

June 2024

DOI

10.1109/CVPRW63382.2024.00339

Included in this dissertation with permission from IEEE.
©2024 IEEE.



Table tennis ball spin estimation with an event camera

Thomas Gossard* Julian Krismer* Andreas Ziegler Jonas Tebbe Andreas Zell
Cognitive Systems Group, University of Tübingen, Germany
thomas.gossard@uni-tuebingen.de

Abstract

Spin plays a pivotal role in ball-based sports. Estimating spin becomes a key skill due to its impact on the ball's trajectory and bouncing behavior. Spin cannot be observed directly, making it inherently challenging to estimate. In table tennis, the combination of high velocity and spin renders traditional low frame rate cameras inadequate for quickly and accurately observing the ball's logo to estimate the spin due to the motion blur. Event cameras do not suffer as much from motion blur, thanks to their high temporal resolution. Moreover, the sparse nature of the event stream solves communication bandwidth limitations many frame cameras face. To the best of our knowledge, we present the first method for table tennis spin estimation using an event camera. We use ordinal time surfaces to track the ball and then isolate the events generated by the logo on the ball. Optical flow is then estimated from the extracted events to infer the ball's spin. We achieved a spin magnitude mean error of 10.7 ± 17.3 rps and a spin axis mean error of $32.9 \pm 38.2^\circ$ in real time for a flying ball.

1. Introduction

Spin is crucial in ball sports, enhancing ball control and adding an element of unpredictability for the opponent. Whether it is topspin in tennis, backspin in table tennis, or spin variations in cricket, it adds complexity to the game. In soccer and baseball, players use spin to curve shots, making it challenging for opponents to predict the ball's trajectory. This makes spin estimation decisive for playing such sports. Thus, it is also a key component for developing autonomous agents and sports analysis systems. However, in most cases, the spin is too fast for low frame rate cameras, which results in severe motion blur. Event-based cameras do not suffer as much from motion blur as standard frame-based cameras. Indeed, event cameras capture logarithmic changes in a scene's brightness with high temporal resolution in the order of μs . Unlike traditional frame-based cam-

*equal contribution
This research was funded by Sony AI.

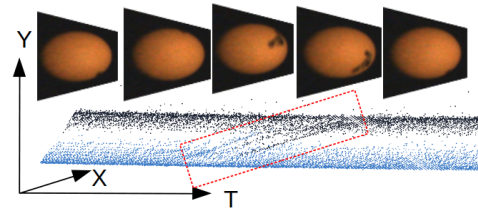


Figure 1. Comparison between the event stream of an event-based camera and captured frames from a frame-based camera. Black and blue dots are, respectively, ON- and OFF-events. The dotted red box highlights the events generated by the spinning logo.

eras, events are only emitted when there is a visual change, offering low latency and reduced bandwidth requirements. This leads to much less information to be transmitted, given a scene with a static background. As a consequence, we do not need to limit the Region of Interest (ROI) of the camera to avoid bandwidth issues. This was indeed a problem in [39], where the full resolution of the camera (1920x1200 pixels) could not be used while maintaining a high enough frame rate (380 fps). Instead, a ROI of 1920x400 pixels was chosen. The ball is thus observed for a shorter time, which decreases the likelihood of observing the ball's logo for low spin values. The event camera can use its full resolution because it does not need to transmit redundant information. The high temporal resolution of event cameras has an additional advantage. Estimating spin using a frame-based camera is constrained by the Nyquist-Shannon sampling theorem, which stipulates that the sampling rate must be at least twice the signal's bandwidth. In other words, we need a camera with a certain frame rate to be able to measure high spin: $\text{fps} > |\omega|/2$, where ω is the ball spin in rps. In Fig. 1, we can see that we only capture two frames where the logo is visible. This makes it extremely difficult to estimate spin accurately. On the other hand, the event camera could continuously capture the movement of the logo. Occasionally, the ball rotates in a manner that causes the logo to consistently appear on the opposite side when captured by a frame-based camera. This is no issue for event-based cam-

eras, as events are generated asynchronously. This overcomes the previously mentioned issue as long as the logo moves in front of the camera. For all the previously mentioned reasons, event cameras are better suited for table tennis ball spin estimation.

Contributions of this work are as follows:

- Tracking the ball in real-time and extracting events generated by the logo.
- A real-time and accurate method for table tennis ball spin estimation.
- An evaluation using a ball spinner to show the potential of the approach and a deployment with a ball thrower to show the approach in a real setup.

2. Related work

2.1. Spin estimation

There are multiple approaches for spin estimation in table tennis. Human players mostly estimate the spin applied to the ball from the motion of the opponent's racket. This can be reproduced by using human pose estimation [26, 31], racket tracking [18], or an inertial measurement unit (IMU) installed in the racket [5, 6]. However, the estimated spin is only an approximation, as there is no information on the racket's bounce dynamics, which are greatly influenced by its rubber type. For a more quantitative spin estimate, there are two main approaches: observing the trajectory or directly observing the ball. Trajectory-based spin estimation [11, 34, 39, 42] makes use of the Magnus effect, which bends the ball's trajectory depending on the magnitude and type of spin applied. Nevertheless, these approaches rely heavily on precise estimations of ball positions. The Magnus effect correlates directly with the level of spin imparted on the ball, complicating the differentiation between the spin-induced curvature of the trajectory and measurement inaccuracies of the ball positions. While [2] showed that the ball's trajectory is insufficient for accurate spin estimation, the estimation of the spin can be further improved by using the bounce behavior of the ball [42].

However, for the most accurate spin estimation, ball observation methods are preferred. Using the blur generated by the logo for spin estimation has been studied in [7] but it does not seem to scale well to higher spin values. Within the ball observation methods, we distinguish two sub-categories: logo-based [22, 39, 43] and pattern-based [16, 23, 35, 37, 38, 41]. The patterns are always visible in contrast to the logo which is barely visible or hidden from a single camera's point of view, making previous logo-based approaches unreliable. For official matches, custom balls are not allowed. This makes logo-based methods necessary for competitive games. Our proposed approach leverages event cameras for ball spin estimation.

All of the previously described methods are not exclu-

sive. They can be combined for better spin estimation. A more recent work [36] combined a logo-based method with a trajectory-based approach. The ambiguity about the spin magnitude stemming from the fact that the Nyquist-Shannon sampling theorem might not be fulfilled was resolved by estimating the ball trajectory for various spin magnitudes and choosing the spin magnitude for which the current ball trajectory fits best to the estimated one.

2.2. Event cameras

Event cameras offer many advantages, such as a high temporal resolution and a high dynamic range. However, they also change the processing paradigm. Most computer vision algorithms developed in the last decades are either ill-adapted to process events or do not make the most out of said advantages. Frames can be reconstructed from events to apply state-of-the-art computer vision algorithms, but this introduces extra latency and does not fully utilize the event cameras' speed.

Tracking fast-moving objects is a perfect use case for event cameras, thanks to their high temporal resolution and less severe motion blur. A lot of research has focused on fusing the data from frame-based and event-based cameras [8–10, 44] to compensate for each other's weaknesses: frames for static or slow moving objects and events for fast-moving objects.

Because an event camera generates only ON/OFF events, new algorithms were developed to distinguish different moving objects in a scene. [14, 15] tackle this issue by segmenting events generated by the background and events generated by independently moving objects with a motion-compensated mean time stamp image. Events are then clustered using DBSCAN. Such methods were used for either obstacle avoidance for drones [14] or for capturing a ball with a quadruped robot [15]. Next to the rather specific motion-compensated mean time stamp image, other common event representations are accumulated event frames and time-surfaces [17]. However, the same object will not be viewed the same way depending on its velocity. This was solved with an ordinal time surface such as TOS [21] and EROS [19]. These ordinal time surfaces capture the edge of objects without any trail of events, independent of the object's velocity. A similar representation could be achieved with accumulated event frames, but only if the accumulation time was tuned to fit the object's velocity.

Feature tracking can also benefit from using event cameras, as shown in [3, 12]. However, because of the logo's small size and rapidly changing shape due to the curve of the spinning ball, such methods are difficult to apply.

Optical flow is an alternative for estimating the logo's speed. There are multiple ways to infer the optical flow. There are Lucas-Kanade [27], plane-fitting [4], and time surface matching [28], just to name a few. However, the

current state-of-the-art is learning-based [20]. The most common architectures are U-Net [30], FireNet [32] and RAFT [40]. The optical flow can be learned in either a supervised [20], semi-supervised [24], or unsupervised [25] fashion. Although more accurate, learning-based optical flow methods require more computational power, which makes achieving real-time performance difficult. For this reason, we favor a model-based approach.

3. Method

Our table tennis ball spin estimation pipeline is divided into three phases. Initially, a ball tracker, outlined in Sec. 3.1, estimates the ball’s position, velocity, and radius. Next, this information is used to extract events generated by the logo, as described in Sec. 3.2. Finally, the ball’s spin is estimated from the extracted events, detailed in Sec. 3.3.

3.1. Ball tracker

We set up a static event camera, observing a primarily static background (a table tennis table). This configuration is beneficial since the flying ball triggers most of the events.

We use the Exponential Reduced Ordinal Surface (EROS) [19] event representation for processing the events. EROS works by having new events decay the values of surrounding pixels by a factor, thus enabling objects to have sharp edges, whatever their velocity. This representation enables continuous and asynchronous updates from the event stream, maximizing the potential of the event camera. The update of EROS depends on a parameter named k_{eros} , which represents the update zone around the event. k_{eros} was hand-tuned for the best circle detection and set to 10. A comparison between the accumulated event frames and EROS can be seen in Fig. 2. One issue with the EROS time

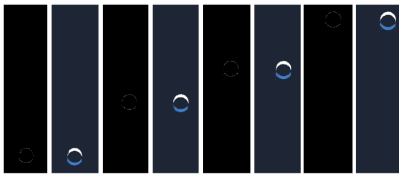


Figure 2. Pairs of representations for the same events. (left) EROS [19] event representations with $k_{eros} = 10$. (right) Accumulated event frames with an accumulation time of $t_{acc} = 2\text{ms}$.

surface is the accumulation of noise. Because the noise is spatially and temporally sparse, it stays in the representation. We thus “clean” the EROS time surface by removing isolated pixels. We do so with a hit-or-miss transform to detect isolated non-zero pixels. Detected isolated pixels are then set to zero. This hit-or-miss transform is performed on

every update of the EROS time surface. For more details, we refer to Sec. 9 in the supplementary material.

We do not use convolutions like in [19] because we also need the radius of the ball for extracting the events generated by the logo as explained in Sec. 3.3. Instead, we apply a Hough circle transform to detect the ball and extract the peak value when the accumulation exceeds a certain threshold.

We added a Kalman filter to improve the ball tracker’s performance. This allows for smoother tracking as well as ball velocity estimation. We also estimate the ball’s radius to reduce the search space of the Hough circle transform for faster processing and for converting distances in pixels to meters. The state of the Kalman filter is defined with the state $\mathbf{x} = [x_b, y_b, \dot{x}_b, \dot{y}_b, r]$ where x_b and y_b are the coordinates of the center of the ball, $\dot{x}_b$ and $\dot{y}_b$ are the ball’s velocity in pixels/s and r is the ball’s radius in pixels. Estimating the ball velocity and radius is relevant for extracting the events generated by the logo as explained in Sec. 3.2. The state transition of the Kalman filter is defined as

$$F = \begin{bmatrix} 1 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1)$$

where dt is the time difference between two Kalman filter updates. The observation vector of the Kalman filter is $\mathbf{z} = [x_{b,\text{hough}}, y_{b,\text{hough}}, r_{\text{hough}}]$ where $x_{b,\text{hough}}$ and $y_{b,\text{hough}}$ are the coordinates of the ball center returned by the Hough circle transform and r_{hough} is the corresponding radius. The observation model of the Kalman filter is defined as

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2)$$

Though the ball’s velocity is not directly observed, it can implicitly be estimated from the sequential measurements of ball positions. The initial variance of the velocities must be set high enough to converge fast enough to the correct velocity. In Fig. 3, we show an example of the Kalman filter tracking the ball. We can observe that the radius of the ball is not constant. This is because the ball can get closer to the camera after being shot.

3.2. Extracting logo events

The ball is tracked accurately using the previously described method. With the Kalman filter, we have an estimate of the ball’s velocity and radius. The ball’s velocity and radius can be used to extract the events generated “inside” the ball, i.e., events generated by the logo. Events that we assume are generated by the logo have to satisfy

$$(y_b(t_e) - y_e)^2 + (x_b(t_e) - x_e)^2 < (r - \text{pad})^2, \quad (3)$$

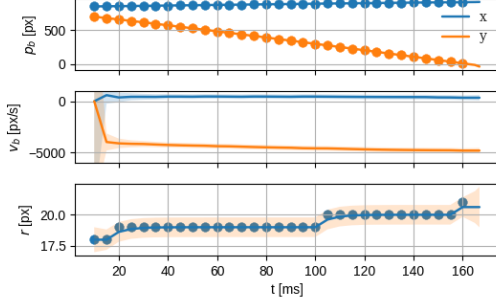


Figure 3. Tracked states of the Kalman filter: Position, $\mathbf{p}_b = [x_b, y_b]$, velocity, $\mathbf{v}_b = [\dot{x}_b, \dot{y}_b]$ and ball radius r . The Kalman filter runs with 200 Hz. The dots indicate measurements from the hough circle transform. The colored band around the estimated values represents the standard deviation.

where t_e is the event's time stamp, x_e the x-position of the event, y_e the y-position of the event, r the radius of the ball and pad is a tolerance to avoid extracting events from the edge of the ball.

For maximal accuracy, the ball's position is continuously updated to the time stamp of the selected event, according to

$$\begin{aligned} \mathbf{p}_b(t_e) &= (x_b(t_e), y_b(t_e)) \\ &= \mathbf{v}_b(t_i) \cdot (t_e - t_i) + \mathbf{p}_b(t_i), \end{aligned} \quad (4)$$

where t_i is the previous time the Kalman filter's state was updated. With this, we can get the events generated by the logo of a flying ball.

3.3. Spin Estimation

Frame-based table tennis spin estimation methods capture successive logo orientations and then regress the spin. Though this method could be translated to event-based cameras, it would not fully exploit their advantages.

Generally, optical flow is a good tool for estimating velocities or, in this case, spin. As mentioned in Sec. 2, there are numerous optical flow methods for event cameras.

We first explain how we obtain the ball's spin ω from the optical flow v_f generated by the logo. From the flow, we can calculate the velocity $\mathbf{v}$ of the point on the ball's surface where the flow is generated

$$\mathbf{v} = \begin{bmatrix} v_{f,x} \\ v_{f,y} \\ \frac{-v_{f,x}e_{r,x} - v_{f,y}e_{r,y}}{e_{r,z}} \end{bmatrix}. \quad (5)$$

We can calculate the z component of $\mathbf{v}$ by knowing that the velocity is tangent to the ball's surface, $\mathbf{e}_r \cdot \mathbf{v} = 0$. This is sketched in Fig. 4. We can then calculate the rotation vector

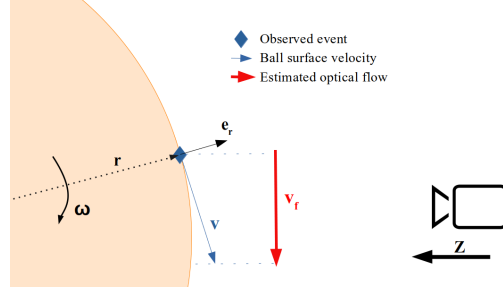


Figure 4. Sketch showing how the spin is calculated from the optical flow.

from the cross-product of the radius and the velocity.

$$\omega = \frac{\mathbf{e}_r \times \mathbf{v}}{\|\mathbf{r}\|}. \quad (6)$$

When working with different optical flow methods, we noticed that they are able to infer flow correctly when the logo is in the center of the ball. But they fail as soon as the logo is on the edge of the ball. This is due to multiple factors. First, table tennis ball logos have multiple edges. As such, events are simultaneously generated at different places of the logo and not in a straight line. This makes identifying successive events generated by the same edge difficult. Secondly, because of the geometry of the ball, the logo's velocity is not constant. There are several parameters to tune the different optical flow algorithms. The main one is the accumulation time, t_{acc} , which aggregates events into an event frame used for optical flow. To obtain the best optical flow results, the t_{acc} should be chosen based on the logo's speed to ensure enough change to observe flow but not so much as to cause motion blur. For a fast, first estimate, we choose t_{acc} as a constant value, based on the median of the expected spin speeds as $t_{acc} = \lceil (\omega/10) \rceil$. From testing, we divided each rotation into 10 time slices because it ensures that even if the true spin speed is higher than we estimated, we still observe the flow multiple times per rotation.

For a better estimate of the spin magnitude and therefore t_{acc} , we also consider a second approach based on the variation of the event rate from the extracted logo events. Indeed, peaks in the event rate indicates when the logo is periodically the most visible. This approach is inspired by [29] and it returns results after a longer observation time but with a higher accuracy. We first apply an exponential moving average (EMA) and subtract the mean event rate. Then, we apply a low pass filter to smooth the curve. After that, we locate all the time stamps at which the sign of this curve transitions from positive to negative values. Since the change in the event rate is periodic and depends on the

ball's spin, as the logo appears and disappears, it can be used to estimate the magnitude of the ball's spin. Figure 5 shows how the sign transitions of the event rates EMA can be used to calculate the magnitude of the spin, by taking the time difference between adjacent time stamps, calculating the mean, and converting this time per rotation into the number of rotations per second.

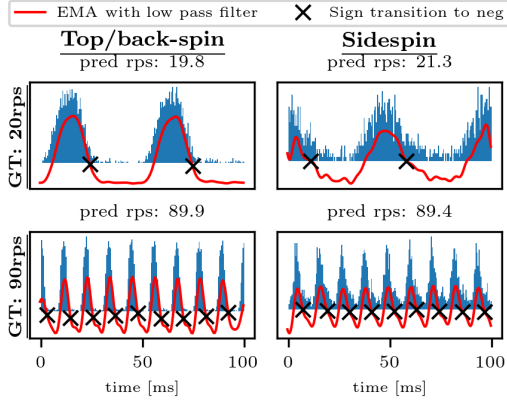


Figure 5. Prediction of the spin magnitude from the event rate. The histograms of the event rate are displayed. We first apply an exponential moving average (EMA) and subtract the mean event rate. Then, we apply a low pass filter to smooth the curve. After that, we locate all the time stamps at which the sign of this curve transitions from positive to negative values.

We can then apply the various optical flow methods for both methods of choosing t_{acc} . The parameters are chosen for each one depending on t_{acc} to return the best results.

For each t_{acc} , several flows and consequently spins are estimated, with the resulting value being the mean of all computed ω . This process can be repeated for each t_{acc} to generate additional spin estimations, which are then averaged collectively.

4. Experiments

We start this section by describing our experimental setup in Sec. 4.1. In Sec. 4.2 we evaluate the ball detector and in Sec. 4.3 we describe our experiment to evaluate the spin estimation of our proposed approach.

4.1. Setup

We use a Prophesee EVK4 event camera with a resolution of 1280x720 pixels for capturing events. The camera is mounted ~ 2 m above the table tennis table, equipped with a 8mm lens. This allows us to cover the whole width of the table, as shown in Fig. 6. The camera's focus was adjusted using the protocol provided by Prophesee [1]. Event-

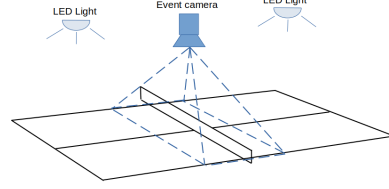


Figure 6. Event camera recording setup

based cameras possess particular settings that differ from standard, frame-based cameras. These were investigated and tuned to generate the most events related to the ball and logo observation while minimizing noise. For more information about the bias settings used with quantitative and qualitative results, please refer to Sec. 6 in the supplementary material. We use LED panels to light the scene uniformly from the ceiling, leading to a luminosity of around 1930 Lux measured on the table. This is to avoid the flickering light from standard lighting powered with 50Hz AC, which triggers events over the entire scene in the event camera, though events generated by the flickering light can also be filtered out with a band-cut filter.

We rely on a custom-made static ball spinner and a ball thrower (Amicus Prime from Butterfly) to generate recordings with known ground truth. The ground truth spin can be calculated from the ball spinner by knowing its relative position to the event camera. We mounted the balls with different logo orientations on the ball spinner to study its impact on the spin estimation, as shown in Fig. 7.



Figure 7. Balls with different logo orientations

For the ball thrower, we rely on SpinDOE [23] to generate ground truth spin, using balls with a custom pattern drawn on them. A frame camera, the Grasshopper3 GS3-U3-23S6C (1900x400), was installed next to the event camera for capturing frames of the balls.

4.2. Ball detection

To evaluate the ball tracker, we recorded multiple balls flying in front of the event camera with speeds ranging from 4 m/s to 12 m/s. The ground truth was generated automatically with blob detection applied on accumulated event frames with a large accumulation time. In Fig. 8, we show

our ball detector’s Mean Absolute Error (MAE) for different ball thrower settings (arbitrary unit). We can see that we have sub-pixel accuracy, which is necessary for extracting the events generated by the ball’s logo.

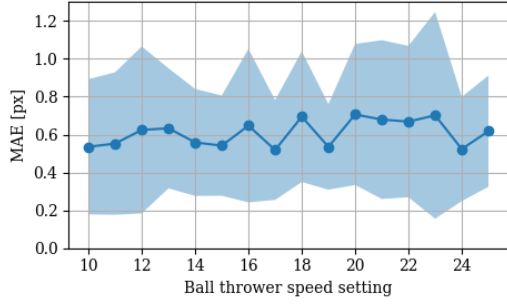


Figure 8. Evaluation of our ball detector for different ball thrower settings (integer values), reporting the MAE in pixels. The colored band around the error represents the standard deviation.

4.3. Spin estimation

We tested our spin estimation method by two different means: with a ball spinner and with a ball thrower. The former has the advantage to give us an extremely accurate and reliable ground truth while the later is closer to real use case. Both benchmarks are still quite similar from the way events generated by the logo are extracted from the flying ball as explained in Sec. 3.2. Indeed, the extracted events’ position is recalculated using the ball position as a reference.

We selected Triplet Matching [33] as the optical flow algorithm. Triplet Matching is the fastest optical flow method we found. It has a sub-millisecond run-time while maintaining competitive accuracy. Moreover, it runs on a CPU, which limits hardware requirements.

We distinguish two types of spin: backspin/topspin and sidespin. Because the ball is observed from the top, backspin, and topspin are almost the same, if only mirrored. With top and backspin, the logo will come into view of the camera and then disappear again. Sidespin balls, on the other hand, will either always have their logo hidden or always visible.

4.3.1 Ball spinner benchmark

We used table tennis balls with different logo orientations to evaluate our proposed spin estimation approach, as shown in Fig. 7. Next to the spin magnitude, our method also estimates the spin axis. The spin magnitude estimation error and the spin axis estimation error with the ball spinner for different ball types are shown in Tab. 1.

Entries with “-” indicate that we did not get any results. For ball 1, the sidespin could not be estimated since the logo and spin axis coincide. The logo of balls 6 and 7 is a rotated version of ball 1; therefore, the same restriction applies.

The errors with the ball spinner for different ball spins are reported in the top part of Fig. 9. As can be seen in the plot, the MAE for the lowest spin magnitude of 10rps is very high. The reason is twofold. First, due to the relatively slow rotation speed, the logo triggers fewer events, which makes it difficult for the optical flow to get accurate flow estimations. Second, with the relatively slow rotation speed, the likelihood of observing the logo over 10ms is relatively low. With 20rps, the MAE is already lower and relatively stable up to 100rps for all four cases except for the sidespin with 10ms observation time. For sidespin with 10ms observation time, the MAE increases linearly with the spin speed. This underlines the difficulty of estimating sidespin.

The MAE of the spin axis estimation errors with the ball spinner for different ball spins are plotted in the bottom graph of Fig. 9. As can be seen in the plot, the spin axis estimation error decreases with higher spin values. This is because higher spin values lead to more events, allowing the optical flow algorithm to achieve higher accuracy.

4.3.2 Ball thrower benchmark

Next to the evaluation with a ball spinner, we also deployed our method in a setup to estimate the spin of flying balls. A ball thrower shot 20 balls. We used velocity settings of 10, 15, 20, and 25 for the ball thrower (respectively approximately 4, 5.5, 7.5 and 9 m/s) and top- and side-spin, with spin strength settings ranging from -5 to 7 . Both our event-based approach and a state-of-the-art frame-based approach [39] were tested on the same observations. The results are listed in Tab. 2. The success rate in the first row is the ratio between the number of successful spin estimations and the total number of ball trajectories. As can be seen, our approach has a success rate slightly above the frame-based approach. Our approach is a bit behind in terms of spin axis and spin magnitude MAE, however, with overlapping error ranges. Given that there is already a considerable body of literature covering frame-based spin estimation methods and our method is the first event-based one, this should show the potential of our approach.

In Fig. 10, we show the spin magnitude MAE on the top and the spin axis MAE on the bottom for a state-of-the-art frame-based approach [39] and our event-based approach. As expected, we see that the frame-based approach has more problems with higher spin settings, while the event-based approach struggles with low spin settings. It is worth noting that though both frame and event cameras have the same FOV, the frame camera boasts a higher resolution ($\sim \times 1.5$). With a resolution similar to the frame camera,

Appendix E Table tennis ball spin estimation with an event camera

	Ball	1	2	3	4	6	7
10ms	Back/Top-spin	30.0 ± 28.7	27.0 ± 27.5	23.36 ± 26.4	22.4 ± 21.9	32.0 ± 27.3	32.5 ± 28.2
		$18.1 \pm 10.4^\circ$	$25.3 \pm 2.0^\circ$	$27.6 \pm 12.7^\circ$	$25.6 \pm 3.9^\circ$	$15.4 \pm 19.6^\circ$	$9.5 \pm 7.2^\circ$
	Sidespin	-	31.0 ± 20.6	30.5 ± 18.8	32.0 ± 23.4	-	-
		-	$38.17 \pm 14.6^\circ$	$40.3 \pm 14.3^\circ$	$42.8 \pm 7.9^\circ$	-	-
100ms	Back/Top-spin	0.8 ± 0.1	0.5 ± 0.1	2.8 ± 2.7	3.7 ± 3.9	0.6 ± 0.1	0.5 ± 0.1
		$12.5 \pm 4.0^\circ$	$25.1 \pm 1.8^\circ$	$25.7 \pm 1.4^\circ$	$25.5 \pm 1.9^\circ$	$10.7 \pm 5.5^\circ$	$6.2 \pm 3.2^\circ$
	Sidespin	4.8 ± 1.8	8.4 ± 1.5	8.0 ± 3.5	6.5 ± 3.2	-	-
		-	$16.7 \pm 9.4^\circ$	$11.6 \pm 13.5^\circ$	$24.0 \pm 10.5^\circ$	-	-

Table 1. Evaluation of our spin estimation with the ball spinner. Our test was performed with balls with different logo positions relative to the spin axis, as shown in Fig. 7 (white rows: MAE for the spin magnitude in rps, green rows: MAE for the spin axis in degrees). 10ms and 100ms indicate the amount of observation time allowed. Entries with “-” indicate that we did not get any results. For ball 1, the sidespin could not be estimated since the logo only rotates around the spin axis of the ball spinner, and there is no sidespin. The logo of balls 6 and 7 is a rotated version of ball 1; therefore, the same restriction applies.

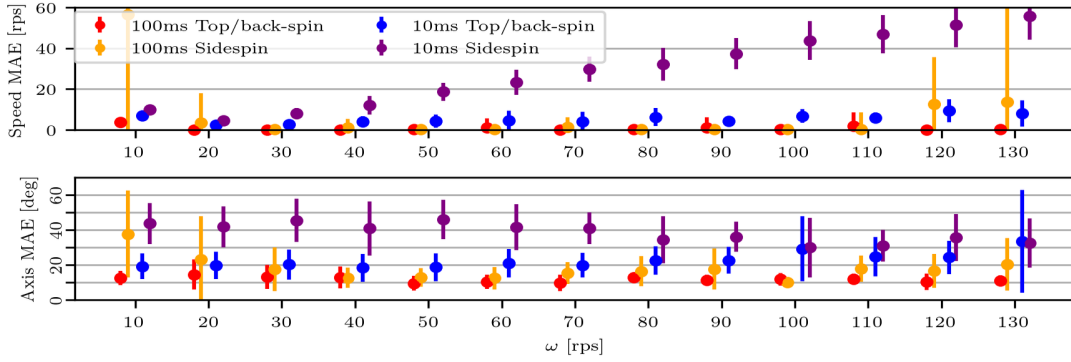


Figure 9. (top) Spin magnitude (speed) error with the ball spinner for different ball spins. (bottom) Spin axis estimation error with the ball spinner for different ball spins. 10ms and 100ms indicate the amount of observation time allowed. Only the events recorded during that time window can be used for the estimation. Backspin/Topsin was evaluated with balls 1, 6, and 7, and sidespin was evaluated with 2, 3, and 4, with an equal number of samples.

	Frame	Event
Success rate	0.79	0.81
Spin axis MAE	$26.6 \pm 33.4^\circ$	$32.9 \pm 38.2^\circ$
Spin magnitude MAE	8.8 ± 15.6 rps	10.7 ± 17.3 rps

Table 2. Benchmark with the flying balls from a ball thrower

the event-based spin estimation would work better. With these complementarity properties, an approach combining an event-based camera and a frame-based camera could be an option.

4.4. Run times

Since our goal is a real time capable spin estimation pipeline, the run-time is another important metric. We used an Intel Core i7-9700 CPU @ 3.00GHz for our experiments,

with all the components written in Python.

From the moment the events are received from the camera, generating the EROS time surface, detecting and tracking the ball, and extracting the logo events takes 12.8 ± 1.2 ms. We run the spin estimation in post-processing and report the ratio between the time it took to process all the events and the duration of the recording. With a ratio of 0.84 ± 0.8 , the spin estimation can process more data than it receives and, therefore, runs in real time.

While it can already be considered capable of being in real time, the setup could be improved. Improving the software design and implementing the software pipeline in C++ instead of Python is expected to boost performance.

4.5. Limitations

While working on the spin estimation for the flying balls from a ball thrower, we noticed fewer events being gen-

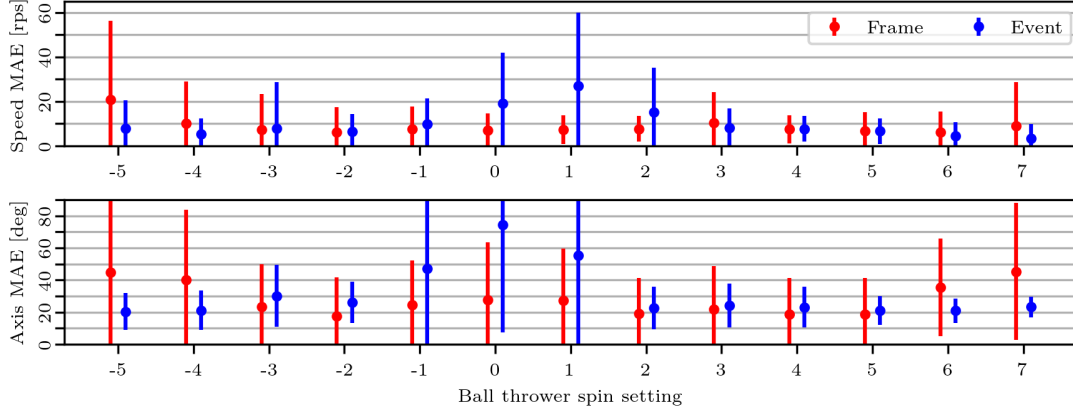


Figure 10. (top) Spin magnitude error with the ball thrower for different ball spins. (bottom) Spin axis estimation error with the ball thrower.

erated compared to the ball spinner setup. This led to an increase in spin estimation failures because there were too few events generated by the logo for the optical flow estimation. Our interpretation of this phenomenon is that pixels respond slower to negative luminosity changes, such as the movement of a logo, after being triggered by a positive change, such as the appearance of the leading edge of a flying ball. This is sometimes referred to as the event camera equivalent of motion blur. Though our method works with the Andro logo displayed in Fig. 11a, it does not generalize to all logos. For example, our spin estimation algorithm does not work with the ball in Fig. 11b since the logo does not generate enough events. This is despite the fact that it works with the ball spinner. We tested different custom logos to determine the causing factor. The logo lines must be thicker than a certain threshold to be detected. We drew lines of different line widths on logoless balls to show this. The ball with the 0.8mm line shown in Fig. 11c did not generate enough events, but the ball with 1.4mm line shown in Fig. 11d worked well with our spin estimation method. As such, only logos with lines of a certain width

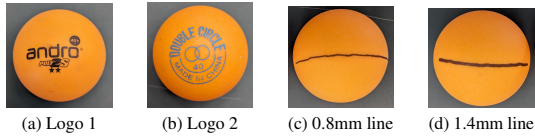


Figure 11. Different ball logos used to estimate the spin of the ball

can be used for our spin estimation with an event camera. Of course, there are different solutions to circumvent this. We could use an event camera with a higher resolution or

a lens with a longer focal length. However, with the latter solution, the camera’s field of view would no longer cover the whole width of table.

We also noticed that the spin estimation has a singularity. Indeed, out of all the balls shown in Fig. 7, there is one logo position, ball 5, for which the spin estimation failed completely, whatever the spin type. Ball 5 does not work for backspin/topspin because the logo is barely visible on the side of the ball. For sidespin, the estimation failed because the rotation axis and the logo coincided. This could potentially be circumvented by using an optical flow algorithm better suited to our specific application.

5. Conclusion

In this work, we presented a spin estimation method for table tennis using optical flow based on the event stream of an event camera. Though this method was implemented and tested for table tennis, it could be generalized to other sports. We were able to compensate for the drawbacks of frame-based cameras with an event-based camera, which allows for more reliable spin estimation.

Although we have shown that our presented approach works for balls on a ball spinner and also for flying balls, depending on the setup, there are still some limitations, notably the slower response of OFF events of pixels after being triggered ON.

For future work, we plan to improve our setup and generate a dataset of event recordings of spinning balls to train a neural network specifically for event-based spin estimation. This would allow us to bypass the optical flow estimation. With such a method, we hope to achieve faster and more accurate spin estimation.

References

- [1] Focus adjustment (c++) - metavision sdk docs 4.5.2 documentation. [5](#)
- [2] Jan Achterhold, Philip Tobuschat, Hao Ma, Dieter B  chler, Michael Muehlebach, and Joerg Stueckler. Black-box vs. gray-box: A case study on learning table tennis ball trajectory prediction with spin and impacts. In *Proceedings of the 5th Annual Learning for Dynamics and Control Conference (LADC)*, pages 878–890. PMLR, 2023. [2](#)
- [3] Ignacio Alzugaray and Margarita Chli. HASTE: multi-hypothesis asynchronous speeded-up tracking of events. In *31st British Machine Vision Conference 2020, BMVC 2020*. BMVA Press, 2020. [2](#)
- [4] Myo Tun Aung, Rodney Teo, and Garrick Orchard. Event-based plane-fitting optical flow for dynamic vision sensors in fpga. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018. [2](#)
- [5] Peter Blank, Julian Ho  bach, Dominik Schuldhaus, and Bjoern M. Eskofier. Sensor-based stroke detection and stroke type classification in table tennis. In *Proceedings of the 2015 ACM International Symposium on Wearable Computers - ISWC '15*. ACM Press, 2015. [2](#)
- [6] Peter Blank, Benjamin H. Groh, and Bjoern M. Eskofier. Ball speed and spin estimation in table tennis using a racket-mounted inertial sensor. In *Proceedings of the 2017 ACM International Symposium on Wearable Computers*. ACM, 2017. [2](#)
- [7] Giacomo Boracchi, Vincenzo Caglioti, and Alessandro Giusti. *Single-Image 3D Reconstruction of Ball Velocity and Spin From Motion Blur - An Experiment in Motion-from-Blur*. SciTePress - Science and Technology Publications, 2008. [2](#)
- [8] Luis A. Camunas-Mesa, Teresa Serrano-Gotarredona, Sio-Hoi Ieng, Ryad Benosman, and Bernabe Linares-Barranco. Event-driven stereo visual tracking algorithm to solve object occlusion. *IEEE Transactions on Neural Networks and Learning Systems*, 29(9):4223–4237, 2018. [2](#)
- [9] William Chamorro, Juan Andrade-Cetto, and Joan Sol  . High speed event camera tracking. In *31st British Machine Vision Conference 2020, BMVC 2020*. BMVA Press, 2020.
- [10] Haosheng Chen, Qiangqiang Wu, Yanjie Liang, Xinbo Gao, and Hanzhi Wang. Asynchronous tracking-by-detection on adaptive time surfaces for event-based object tracking. In *Proceedings of the 27th ACM International Conference on Multimedia*. ACM, 2019. [2](#)
- [11] Xiaopeng Chen, Ye Tian, Qiang Huang, Weimin Zhang, and Zhangguo Yu. Dynamic model based ball trajectory prediction for a robot ping-pong player. In *2010 IEEE International Conference on Robotics and Biomimetics*. IEEE, 2010. [2](#)
- [12] Philippe Chibierre, Etienne Perot, Amos Sironi, and Vincent Lepetit. Detecting stable keypoints from events through image gradient prediction. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2021. [2](#)
- [13] Tobi Delbruck, Rui Graca, and Marcin Paluch. Feedback control of event cameras. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2021. [1](#)
- [14] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, 5(40), 2020. [2](#)
- [15] Benedek Forrai, Takahiro Miki, Daniel Gehrig, Marco Hutter, and Davide Scaramuzza. Event-based agile object catching with a quadrupedal robot. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023. [2](#)
- [16] Sho Furuno, Kazuyuki Kobayashi, Tomoyuki Okubo, and Yosuke Kurihara. A study on spin-rate measurement using a uniquely marked moving ball. In *2009 ICCAS-SICE*, pages 3439–3442. IEEE, 2009. [2](#)
- [17] Guillermo Gallego, Tobi Delbr  ck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, J  rg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, 2022. [2](#)
- [18] Yapeng Gao, Jonas Tebbe, and Andreas Zell. *Robust Stroke Recognition via Vision and IMU in Robotic Table Tennis*, page 379–390. Springer International Publishing, 2021. [2](#)
- [19] Luna Gava, Marco Monforte, Massimiliano Iacono, Chiara Bartolozzi, and Arren Glover. Puck: Parallel surface and convolution-kernel tracking for event-based cameras, 2022. [2](#), [3](#)
- [20] Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. Dense continuous-time optical flow from event cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–12, 2024. [3](#)
- [21] Arren Glover, Aiko Dinale, Leandro De Souza Rosa, Simeon Bamford, and Chiara Bartolozzi. luvharris: A practical corner detector for event-cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):10087–10098, 2022. [2](#)
- [22] Jared Glover and Leslie Pack Kaelbling. Tracking the spin on a ping pong ball with the quaternion bingham filter. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014. [2](#)
- [23] Thomas Gossard, Jonas Tebbe, Andreas Ziegler, and Andreas Zell. Spindoe: A ball spin estimation method for table tennis robot. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023. [2](#), [5](#)
- [24] Woobin Im, Sebin Lee, and Sung-Eui Yoon. *Semi-supervised Learning of Optical Flow by Flow Supervisor*, page 302–318. Springer Nature Switzerland, 2022. [3](#)
- [25] Rico Jonschkowski, Austin Stone, Jonathan T. Barron, Ariel Gordon, Kurt Konolige, and Anelia Angelova. *What Matters in Unsupervised Optical Flow*, page 557–572. Springer International Publishing, 2020. [3](#)
- [26] Kaustubh Milind Kulkarni and Sucheth Shenoy. Table tennis stroke recognition using two-dimensional human pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 4576–4584, 2021. [2](#)
- [27] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (darpa).

-
- In *Proceedings of DARPA Image Understanding Workshop (IUW '81)*, pages 121 – 130, 1981. 2
- [28] Jun Nagata, Yusuke Sekikawa, and Yoshimitsu Aoki. Optical flow estimation by matching time surface with event-based cameras. *Sensors*, 21(4):1150, 2021. 2
- [29] Bernd Pfrommer. Frequency cam: Imaging periodic signals in real-time, 2022. 4
- [30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*, page 234–241. Springer International Publishing, 2015. 3
- [31] Soichiro Sato and Masaki Aono. Leveraging human pose estimation model for stroke classification in table tennis. In *Working Notes Proceedings of the MediaEval 2020 Workshop, Online, 14-15 December 2020*. CEUR-WS.org, 2020. 2
- [32] Cedric Scheerlinck, Henri Rebecq, Daniel Gehrig, Nick Barnes, Robert E. Mahony, and Davide Scaramuzza. Fast image reconstruction with an event camera. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2020. 3
- [33] Shintaro Shiba, Yoshimitsu Aoki, and Guillermo Gallego. Fast event-based optical flow estimation by triplet matching. *IEEE Signal Processing Letters*, 29:2712–2716, 2022. 6
- [34] Hu Su, Zaojun Fang, De Xu, and Min Tan. Trajectory prediction of spinning ball based on fuzzy filtering and local modeling for robotic ping-pong player. *IEEE Transactions on Instrumentation and Measurement*, 62(11):2890–2900, 2013. 2
- [35] Alexander Szep. Measuring ball spin in monocular video. In *Proc. 16th Comput. Vis. Winter Workshop*, pages 83–89. Citeseer, 2011. 2
- [36] Sho Tamaki, Satoshi Yamagata, and Sachiko Hashizume. Spin measurement system for table tennis balls based on asynchronous non-high-speed cameras. *International Journal of Computer Science in Sport*, 23(1):37–53, 2024. 2
- [37] Toru Tamaki, Takahiko Sugino, and Masanobu Yamamoto. Measuring ball spin by image registration. *Proc. 10th Frontiers of Computer Vision*, pages 269–274, 2004. 2
- [38] Toru Tamaki, Haoming Wang, Bisser Raychev, Kazufumi Kaneda, and Yukihiro Ushiyama. Estimating the spin of a table tennis ball using inverse compositional image alignment. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012. 2
- [39] Jonas Tebbe, Lukas Klamt, Yapeng Gao, and Andreas Zell. Spin detection in robotic table tennis. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020. 1, 2, 6
- [40] Zachary Teed and Jia Deng. *RAFT: Recurrent All-Pairs Field Transforms for Optical Flow*, page 402–419. Springer International Publishing, 2020. 3
- [41] Christian Theobalt, Irene Albrecht, Jörg Haber, Marcus Magnor, and Hans-Peter Seidel. Pitching a baseball: tracking high-speed motion with multi-exposure images. *ACM Transactions on Graphics*, 23(3):540–547, 2004. 2
- [42] Yuxin Wang, Zhiyong Sun, Yongle Luo, Haibo Zhang, Wen Zhang, Kun Dong, Qiyu He, Qiang Zhang, Erkang Cheng, and Bo Song. A novel trajectory-based ball spin estimation method for table tennis robot. *IEEE Transactions on Industrial Electronics*, page 1–11, 2024. 2
- [43] Yifeng Zhang, Rong Xiong, Yongsheng Zhao, and Jianguo Wang. Real-time spin estimation of ping-pong ball using its natural brand. *IEEE Transactions on Instrumentation and Measurement*, 64(8):2280–2290, 2015. 2
- [44] Yabin Zhu, Xiao Wang, Chenglong Li, Bo Jiang, Lin Zhu, Zhixiang Huang, Yonghong Tian, and Jin Tang. Crsot: Cross-resolution object tracking using unaligned frame and event cameras. *arXiv preprint arXiv: 2401.02826*, 2024. 2

Appendix F

An Event-Based Perception Pipeline for a Table Tennis Robot

Authors

Andreas Ziegler, Thomas Gossard, Arren Glover, and Andreas Zell

To be submitted to

2025 IEEE Robotics and Automation Letters (RA-L 2025)

Time of submission

DOI

An Event-Based Perception Pipeline for a Table Tennis Robot

Andreas Ziegler¹, Thomas Gossard¹, Arren Glover², and Andreas Zell¹

Abstract—Table tennis robots gained traction over the last years and have become a popular research challenge for control and perception algorithms. Fast and accurate ball detection is crucial for enabling a robotic arm to rally the ball back successfully. So far, most table tennis robots use conventional, frame-based cameras for the perception pipeline. However, frame-based cameras suffer from motion blur if the frame rate is not high enough for fast-moving objects. Event-based cameras, on the other hand, do not have this drawback since pixels report changes in intensity asynchronously and independently, leading to an event stream with a temporal resolution on the order of μs . To the best of our knowledge, we present the first real-time perception pipeline for a table tennis robot that uses only event-based cameras. We show that compared to a frame-based pipeline, event-based perception pipelines have an update rate which is an order of magnitude higher. This is beneficial for the estimation and prediction of the ball’s position, velocity, and spin, resulting in lower mean errors and uncertainties. These improvements are an advantage for the robot control, which has to be fast, given the short time a table tennis ball is flying until the robot has to hit back.

Index Terms—Table Tennis Robot, Event Camera, Event-Based Computer Vision, Object Detection

SUPPLEMENTARY MATERIAL

Additional resources are available at: <https://cogsys-tuebingen.github.io/event-based-table-tennis>

I. INTRODUCTION

IN recent years, robotic table tennis has become a popular research challenge for control and perception algorithms. While not yet able to compete with professional players, table tennis robots are an exciting research environment to bring perception and control algorithms towards their limits [1] [2]. Fast and accurate ball detection is a crucial perception task for a table tennis robot system. So far, most research uses frame-based cameras together with a Convolutional Neural Network (CNN) based ball detection or a classical computer vision approach [3] [4] [5] [6].

While frame-based cameras are the de-facto standard visual perception sensor, they do suffer from motion blur if the camera’s frame rate is not high enough for fast-moving objects. One way to counteract this limitation is by using cameras with a higher frame rate. However, a higher frame rate leads to more overall data that needs to be processed, increasing the computational resources required. Therefore, the bottleneck shifts

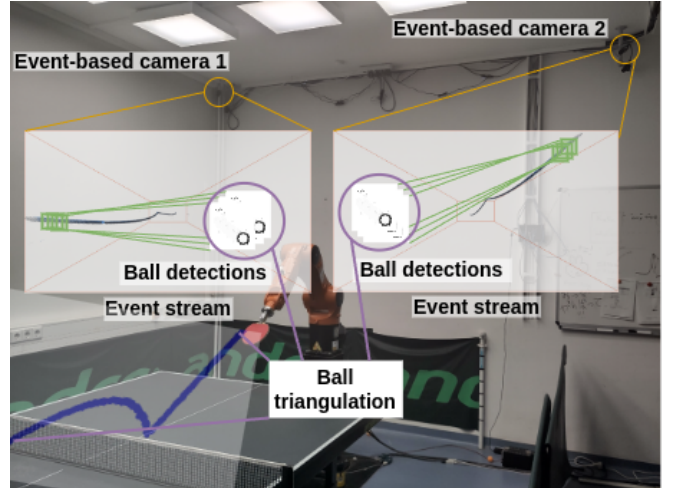


Fig. 1. Background: The industrial robot arm of our table tennis robot setup for which the proposed perception pipeline is designed. The two event-based cameras, indicated with orange circles, are mounted on the ceiling. Foreground: The event streams of the two event-based cameras with detected balls on the EROS event surface, visualized in green, the triangulation process (using the ball detections from both cameras and calculating the 3D position given the camera calibration), indicated in violet, and the resulting triangulated 3D trajectory, shown in blue.

from the sensor to the algorithm. Another way to approach this challenge is to make use of event-based cameras, also known as dynamic vision sensors (DVS) [7]. Event-based cameras report logarithmic changes in intensity if the brightness change exceeds a specified threshold, asynchronously and for every pixel independently, with so-called *events*. The asynchronous event-based nature of event-based cameras enables a high temporal resolution, in the order of μs , a low latency, and a high dynamic range, making them particularly suitable for capturing fast-moving objects [8] [9] [10] and scenes with a high dynamic range [11] [12]. Events are tuples of the form $\langle x, y, t, p \rangle$ with the location of the event (x, y) , the time stamp t and the polarity of the event $p \in \{-1, 1\}$, indicating if the pixel got darker or brighter. Since this data format is fundamentally different from the frames of frame-based cameras, new algorithms are required to make use of the event-based camera’s advantages.

To the best of our knowledge, we present the first real-time perception pipeline for a table tennis robot running entirely with event-based cameras and using their event stream as input. Thus, we address the fundamental task of ball detection in the perception pipeline of a table tennis robot but leave out the robot control part since it is out of scope for this work.

¹Andreas Ziegler, Thomas Gossard, and Andreas Zell are with the University of Tübingen {andreas.ziegler, thomas.gossard, andreas.zell}@uni-tuebingen.de ²Arren Glover is with the Istituto Italiano di Tecnologia, Italy
This research was partially funded by Sony AI.

In event-based computer vision, there is a trade-off between processing single events and batches of events. While a single event does not contain enough information to be useful, processing it is usually fast. On the other hand, event batches contain enough information, but batching introduces a latency, and processing batches tends to be more time-consuming compared to single events. Additionally, using event batches introduces the need to adjust the size of the batch based on the velocity of the perceived objects. Fast moving objects will generate more events than slow moving objects in the same amount of time. If the velocity of objects varies, batching does not create consistent representations.

We maintain a low latency while increasing observed information by processing the incoming events in one thread on an *event-by-event* basis and perform the more compute intensive perception task in a second thread *as-fast-as-possible*, following previous work [13] [14]. This allows the algorithm to make use of event history and, therefore, preserve all the information contained in the event stream.

An event representation that supports such an *event-by-event* update, without the need to recompute the whole representation, is the EROS event representation, used in [14]. We use EROS in combination with a fast circle detector for the ball detection in our perception pipeline.

Contributions of this work are as follows:

- We present the first real-time event-based perception pipeline for a table tennis robot
- With our fast ball detector, our event-based perception pipeline achieves an order of magnitude more position estimates of the ball compared to frame-based pipelines
- We show that this increased position update rate leads to lower uncertainties of the estimated positions, velocities, and spin of the ball's trajectory, which is beneficial for the robot control
- We make the developed perception pipeline publicly accessible on our project page

II. RELATED WORK

In this related work, we will first give an overview of the latest research around table tennis robots in Section II-A. Next, we cover event-based object detection in Section II-B and event representations in Section II-C.

A. Table Tennis Robots

Ever since Billingsley initiated a robot table tennis competition in 1983 [15], robotic table tennis has been a popular tool for research in computer vision and robot control. A table tennis robot using frame-based cameras with a CNN-based ball detection was presented in [5]. In [16], the authors have designed a completely new pneumatic robot arm able to attain very high-end-effector speeds. The authors in [17] introduced a table tennis robot system but focused on trajectory prediction and hitting velocity control. Unfortunately, they do not explain their vision system in detail.

In [4], Google DeepMind presented their table tennis robot system, including perception, planning, and robot control. In a recent update [2], the same authors have shown that

while not yet able to compete with professional players, table tennis robots can compete with amateurs and intermediate-level players. Table tennis robots are also widely used as a use case for reinforcement learning, e.g., in [4] [6] [18] [19].

All of the above table tennis robot systems use frame-based cameras for their perception system and do not make use of event-based cameras. In this work, we developed a perception pipeline for a system similar to [3], using only event-based cameras. We show the advantages of event-based perception pipelines compared to a perception pipeline using conventional, frame-based cameras.

After the ball detection, the next step in a table tennis robot system is the prediction of the ball's trajectory. Given the observation of the positions of the ball until the current time, the robot system needs to predict the ball's future trajectory to plan the hitting stroke of the robot. Even for humans, this prediction is difficult and requires years to get a good estimation of balls with heavy spin. This is in particular due to the difficulty of measuring spin [3].

Next to the more traditional approaches of curve fitting and an Extended Kalman Filter (EKF) in [3], a GRU-based learning approach was used in [20]. In [21], the authors use an EKF and learn the parameters of the dynamics model outperforming two black-box approaches. All these works state the importance of accurate spin and a low spin uncertainty.

B. Event-Based Object Detection

Event-based object detection can be subdivided into model-based and learning-based methods. Most learning-based methods make use of deep learning architectures to learn object detection tasks given a training dataset. In [11], Prophesee presented the first event-based object detector that does not use an intermediate gray scale representation. Recurrent layers are used to take the temporal information of the event data into account. An event-based convolutional layer (e-conv) and an event-based max pooling layer (e-max-pool) were introduced in [22]. Inference time was accelerated using a sparse update method in which only hidden and output neurons changed by incoming events were processed. To make better use of the data structure of the events stream, Graph Neural Networks (GNNs) were used in [23]. The incoming events iteratively update the graph rather than fully reconstructing it at each time-step, preserving the asynchronous nature of event-based cameras. Also, more recent network architectures such as Vision Transformers (ViTs) were introduced for event data [24]. While learning-based methods tend to achieve higher accuracy, their inference time is often longer than the runtime of model-based methods.

An early model-based approach for object detection made use of the Hough transform to track objects with circular structure [25]. In later work, the same authors implemented a particle filter with an observation heuristic, which can be adjusted depending on the object to track [26]. In [10] and [27], events are clustered using DBSCAN to differentiate between the background and independently moving objects with a motion-compensated mean time stamp image. In our setup, the cameras are static, and therefore, motion compensation, which tends to be computationally heavy, is not needed.

Since a fast runtime is essential in our table tennis robot setup, we use a model-based approach with an event representation that allows *event-by-event* updates. Event-based cameras produce a lower-latency ball trajectory prediction as information is available in the system earlier than traditional cameras [8].

C. Event Representation

Depending on the algorithm, events are either consumed *event-by-event* or in a *batch-of-events*. The minority of work does *event-by-event* processing, mostly filter-based approaches, like [28], and Spiking Neural Networks (SNNs), like [29]. If events are processed in batches, there is a variety of choices on how to represent the information present in the event data. For the sake of completeness, we cover the relevant ones here, where we take the definition from [7].

The simplest form is event packets, where events in a spatio-temporal neighborhood are processed together. In event packets, the precise time stamp and polarity information is retained. The next, almost obvious choice is the 2D histogram or event frame. The events in a time window are converted in a simple way into an image/frame that can be fed to image-based computer vision algorithms. The temporal information present in event data is used in the so-called time surface (TS). In a time surface, a 2D map, each pixel stores a single time value. Therefore, the “intensity” of the converted image is a function of the motion history at that location, with larger values corresponding to a more recent motion. The concept of TS was further improved in [30] by introducing a hierarchical representation. In [31], the authors present Histograms of Averaged Time Surfaces (HATS), which is less sensitive to noise and non-idealities of event-based cameras.

Instead of using hand-crafted event representations, the authors in [32] introduced a combination of filters and convolutions, which can be learned in an end-to-end fashion when used in Neural Networks (NNs).

In [?], the authors use a speed invariant time surface for learning to detect corner points with a simple random forest, showing the benefits of a speed invariant representation. A more recent event representation is the threshold-ordinal surface (TOS), introduced in [13]. TOS is designed to accumulate visual data, so it is compatible with conventional image processing algorithms like, e.g., circle or corner detection. The representation provides a coherent and bound spatial representation of the asynchronous events, partially maintaining the information about their temporal order and attempting to capture the most up-to-date position of edges in the scene. In this work, we make use of an improved version of TOS, the Exponential Reduced Ordinal Surface (EROS), used in [14]. EROS can be used at any given point in time by downstream processing, with values between 0 and 255.

III. METHODOLOGY

In this part, we introduce our event-based perception pipeline. We start with a description of the setup in Section III-A. Then, we give an overview of the perception pipeline in Section III-B. In Section III-C, we explain the

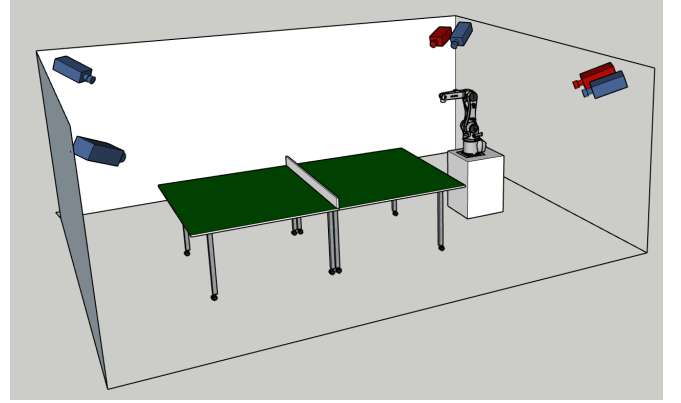


Fig. 2. Our camera setup consists of four frame-based cameras (in blue) and two event-based cameras (in red) with baselines of 3m to 5m. Only the event-based cameras are used in this work. Schematic is up to scale.

event-by-event update of our approach. In the last part, the *as-fast-as-possible* ball detection is covered in Section III-D.

A. Setup

We used a table tennis robot setup similar to [3], but extended it with two event-based cameras. This camera system consists of two hardware-synchronized Prophesee EVK4 event-based cameras (1280x720 pixels). Camera bias settings for the event-based cameras were configured to minimize noise and so that the flying ball would cause most events. Additionally, the Event Trail Filter (STC and Trail)¹ were used to cancel redundant information. The filter considers successive events of the same polarity generated in a short time as a burst and will remove the second event of a burst. The whole table tennis robot system is visualized in Fig. 2. To calibrate this event-based camera system, we used the wand-based calibration approach introduced in [33]. A Butterfly Amicus Prime ball gun with default speed settings (4m/s) was used to shoot the table tennis balls from the opposite side of the robot arm.

For all the software components of our perception pipeline, we used ROS2 [34]. We used the `metavision_driver`² as ROS2 driver for the event-based cameras.

B. The Perception Pipeline

Inspired by [13], we use the best of processing events *event-by-event* and in *batches-of-events*. We do this by running two threads simultaneously. In the first thread, visualized in Fig. 3, our perception pipeline processes the incoming events and updates the EROS event surface *event-by-event*. In the second thread, shown in Fig. 4, the ball detection runs *as-fast-as-possible*. In the next two sections, each of the two parts is described in more detail.

¹<https://docs.prophesee.ai/stable/hw/manuals/esp.html?highlight=stc#event-trail-filter-stc-trail>

²https://github.com/ros-event-camera/metavision_driver

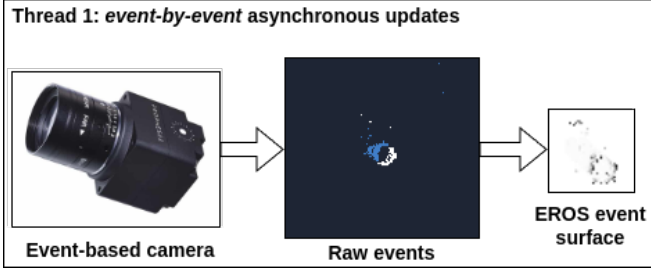


Fig. 3. The first thread of our event-based perception pipeline. We process the incoming event and update the EROS event surface *event-by-event*.

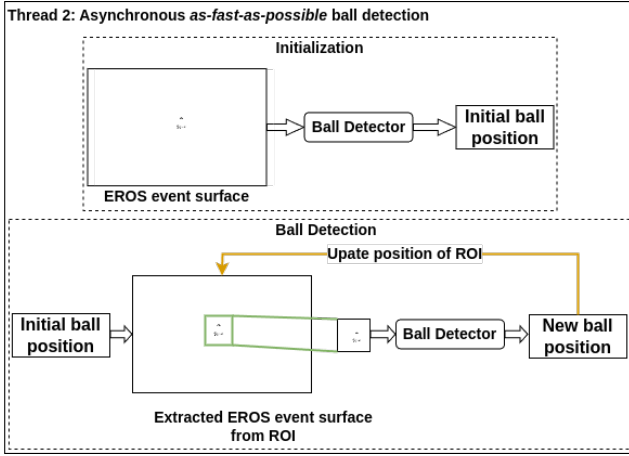


Fig. 4. The second thread of our event-based perception pipeline. We initialize the ROI of the ball and, afterward, detect the ball within the ROI *as-fast-as-possible*.

C. Event-by-event update: EROS Surface

Our perception pipeline uses the asynchronously incoming events to update the EROS event representation *event-by-event*. The EROS event surface allows the decoupling of the asynchronous, high-temporal resolution event stream and the slower processing algorithm.

EROS, with its update step defined in Algorithm 1, provides a coherent and bound spatial representation of the asynchronous events, partially maintaining the information about their temporal order and attempts to capture the most up-to-date position of edges in the scene [13]. Figure 5a shows the input events over time for an example scene. The corresponding EROS event surface is shown in Fig. 5b. The EROS event surface of a flying table tennis ball, used in our application, is visualized in Fig. 6a.

The process time of the EROS update depends on only one parameter, k_{EROS} , which corresponds to the update region size around each event at position (v_x, v_y) . In this work, we use a k_{EROS} of 10. Since the EROS update is a computationally cheap operation, the update can keep up with the incoming event stream up to approximately 10 Mevents/s [14].

D. As-fast-as-possible computation: Ball Detection

As previously mentioned, we use the EROS surface as an event representation. The EROS surface can be sampled at any

Algorithm 1 Event-by-event EROS update

Require: $d = 0.3^{1.0/k_{EROS}}$, k_{EROS} , (v_x, v_y)

for $x = v_x - k_{EROS} : v_x + k_{EROS}$ **do**

for $y = v_y - k_{EROS} : v_y + k_{EROS}$ **do**

$EROS_{xy} \leftarrow EROS_{xy} \cdot d$

end for

end for

$EROS_{v_x v_y} \leftarrow 255$



Fig. 5. Example surface using the EROS algorithm to enable velocity-independent representation without temporal parameter tuning: (a) the input events over time for an example scene, (b) a visualization of the EROS surface after integrating event information over the entire time period of the dataset.

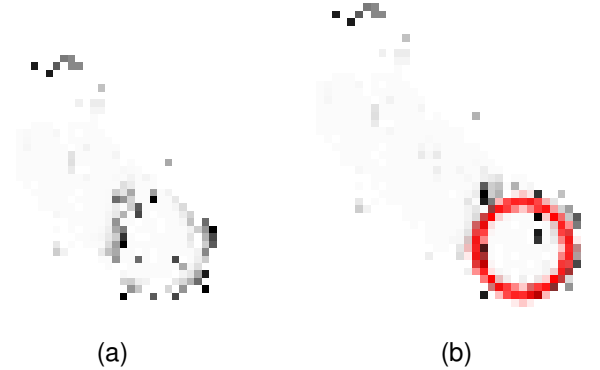


Fig. 6. (a) A visualization of the EROS surface on the data captured for the perception pipeline of the table tennis robot. (b) The detected circle from the Hough-based circle detector overlaid.

point in time with a temporal resolution of the event input and used as a “gray image”, with values between 0 and 255.

Our ball detection is performed on the latest EROS surface, using a fast C++ implementation of a Hough-based circle detector, with an output similar to *HoughCircles()* in OpenCV [35]. The velocity independence of EROS is critical for the Hough transform to succeed as blurred edges due to motion incorrectly bias Hough transform results. An example of a detected ball is shown in Fig. 6b.

Ball detection is done in two modes: an initialization using the whole resolution and within a Region of Interest (ROI) afterward. Since the ball does not move far from one detection to the next, we use the previous ball position as the center of the following ROI. Restricting ball detections to this ROI improves the runtime further, as shown in our experiments in Section IV-C.

TABLE I

OVERALL RESULTS. * SINCE THE UPDATES OF THE FRAME-BASED BASELINE ARE SYNCHRONOUS, IN COMPARISON TO THE ASYNCHRONOUS EVENT-BASED PIPELINES, WE REPORT THE RESULT IN HZ. $\triangle$ THE AUTHORS IN [3] DID NOT PROVIDE IOU RESULTS. $\dagger$ THE EVENT-BASED MEDIAN FILTER BASELINE ONLY PROVIDES THE POSITION, AND THEREFORE, IT IS NOT POSSIBLE TO CALCULATE THE IOU.

Method	Update rate [updates/s]	Average run time (initialization) [ms]	Error [pixels]	IoU
Frame-based baseline [3]	149.0Hz*	0.8 (2.2)	1.32	$\triangle$
Event-based median filter baseline	$(1.38 \pm 0.12) \times 10^4$	0.000 08 (1.1)	2.90 ± 1.60	$\dagger$
Event-based particle filter baseline [26]	$(3.49 \pm 0.40) \times 10^3$	0.7 (1.0)	2.93 ± 1.13	0.55 ± 0.08
Proposed event-based pipeline	$(4.14 \pm 0.10) \times 10^3$	0.1 (0.66)	1.34 ± 0.79	0.78 ± 0.11

IV. EXPERIMENTS AND RESULTS

Fast and accurate ball detection is crucial for enabling a table tennis robot to rally a ball back successfully. The detected ball positions in both cameras need to be accurate so that the triangulated 3D ball positions are precise as well. The speed and update rate of the ball detections are crucial since table tennis is a fast-paced game, and there is only between 0.1s and 1s time from the start of the ball's trajectory until the ball hits the robot's racket. Accurate velocity and spin value estimates with a low uncertainty can additionally help with path planning and control of a robot arm. Therefore, we designed our experiments to evaluate the mentioned key metrics for a perception pipeline used in a table tennis robot.

We start this experimental section by describing the baselines used in Section IV-A. Then, in our first experiment in Section IV-B, we measure the **accuracy** of the different perception pipelines. Next, we measure and compare the **run time** of the perception pipelines in Section IV-C. Besides the run times, we are also interested in the **update rate** to know how frequently the pipelines report the position of the detected ball. This is evaluated in Section IV-D.

We designed the proposed perception pipeline for a table tennis robot system, and therefore, it is crucial to highlight the benefits of the presented approach for the whole robot setup. After the ball detection, the next step in a table tennis robot system is the prediction of the ball's trajectory. In Section IV-E, we compare the trajectory prediction using the ball's 3D position from the frame-based and our proposed event-based system.

A. The Baselines

In the following experiments, we compared our proposed perception pipeline with three other approaches that serve us as baselines. The first is the frame-based system presented in [3]. A first event-based perception baseline is a median filter approach, similar to [36]. The second event-based perception baseline is a particle filter based on [26].

a) *Frame-based pipeline*: Since the majority of the table tennis robot literature uses frame-based cameras for ball detection, we want to compare our proposed event-based method with one that uses only frame-based cameras. While CNN based ball detectors tend to have a slightly higher accuracy, they also tend to be slower than classical ball detectors, using more conventional image processing techniques.

We used the pipeline presented in [3], which has state-of-the-art performance and used a similar setup. In the first step, the method uses the difference between the current frame and

one from the past to avoid being affected by static objects. Afterward, metrics like the aspect ratio, circularity, and size are used to filter out outliers. Since the frame-based cameras run at 149fps, the update rate is 149Hz with a processing latency of around 1ms.

b) *Event-based median filter*: Having static event-based cameras and assuming a static background, one of the simplest approaches to detecting a flying ball is to use the median of the events' positions in a time window. A similar approach, called event clustering, was already used in [36]. This approach will fail if other moving objects in the scene trigger events.

To prevent this failure case to some degree, we used a ROI, in which the median filter will consider events. We implemented an initialization step to set the position of the first ROI. We do this initialization by using a blob detector on accumulated event frames, which we get from the *event_camera_renderer*³ ROS2 node. Later on, the ROI is set to the position of the latest detected ball.

c) *Event-based particle filter*: A more sophisticated, state-of-the-art approach, presented in [26], uses a particle filter. The particle filter not only estimates the ball's position and radius but also the time window for the next step. This allows the approach to adapt to the speed of the moving ball and, therefore, to the difference in the resulting event rate. While we started from the original implementation of [26], we adjusted the implementation and the parameters to work with the fast movement of our flying table tennis balls.

For this particle filter approach, we use the same initialization step as for the median filter described above.

B. Accuracy

To measure the accuracy of the event-based perception pipelines, we proceeded as follows. We compared the 2D position estimates of all the approaches from multiple ball trajectories with ground truth positions. To get ground truth 2D positions, we used a circle detector on reconstructed frames and manually removed inaccurate detections. We used e2vid [37] for the reconstructed frames with our event stream. This resulted in 286 ground truth 2D positions.

The 2D ball detections of the different event-based vision pipelines will not have the same time stamps due to the event data's asynchronous nature and the different approaches' working principles. To account for this, we linearly interpolated the 2D positions to match with the time stamps of the ground truth positions. Given the interpolated 2D positions and the ground truth positions, we calculated the pixel error

³https://github.com/ros-event-camera/event_camera_renderer

between them. The results are listed in the column “Error” in Table I.

As can be seen, the frame-based pipeline does achieve the lowest pixel error with 1.32 pixels, although the proposed event-based perception pipeline with an error of 1.34 pixels is very close. The event-based baselines have an error of 2.90 and 2.93 pixels, respectively, an error more than double the one of the proposed pipeline. The proposed pipeline also has a lower standard deviation (± 0.79 pixels) than the event-based baselines (± 1.60 and ± 1.13 pixels), indicating more stable 2D ball detections.

C. Runtime

To measure the run times of the ball detection pipelines and their components, we used the time measurement functionality of the C++ standard library. We report the mean and standard deviation of two ball trajectories. Since the different ball detection pipelines have various software components, we list the run times for every pipeline separately.

The results of the frame-based baseline are listed in Table II. In Table III, we list the run times of the event-based median filter approach. The results of the event-based particle filter approach are listed in Table IV. For our proposed event-based pipeline, we list the run times in Table V. The overall average run times are listed in Table I.

TABLE II
RUN TIMES OF THE FRAME-BASED BASELINE [3]

Module	Runtime [μ s]
Ball detection (full resolution)	2.2000×10^3
Ball detection (ROI)	800.0

TABLE III
RUN TIMES OF THE EVENT-BASED MEDIAN FILTER BASELINE

Module	Runtime [μ s]
Initialization	$(1.06 \pm 0.19) \times 10^3$
Update	0.08 ± 0.62

TABLE IV
RUN TIMES OF THE EVENT-BASED PARTICLE FILTER BASELINE [26]

Module	Runtime [μ s]
Initialization	$(1.05 \pm 0.20) \times 10^3$
Fetch events	5 ± 11
Particle filter update	700 ± 730
Maximum likelihood calculation	0.19 ± 0.42

TABLE V
RUN TIMES OF OUR PROPOSED EVENT-BASED PIPELINE

Module	Runtime [μ s]
Fetch EROS surface	0.5 ± 1.3
Ball detection (full resolution)	656 ± 38
Ball detection (ROI)	91 ± 18

As can be seen in Table I, the event-based particle filter approach has a similar average run time as the frame-based

approach. In contrast, the other two event-based approaches are significantly faster. During the initialization, the event-based approaches have an average run time of at most half of the frame-based baseline.

As previously mentioned, the median- and particle-filter share the same initialization schema, and therefore, both have an average initialization run time of around 1.1ms. With an average update run time of 0.08μ s, the event-based median filter has the lowest run time after the initialization.

For the event-based particle filter method, the update step is, with around 700μ s, the most costly component. Fetching the events (5μ s) and the maximum likelihood calculation (0.19μ s) is negligible compared to the update step.

In our proposed event-based perception pipeline, the ball detection takes an average of 656μ s during the initialization and 91μ s in the ROI. This is a bigger reduction compared to the frame-based baseline. Fetching the EROS time surface is with 0.5μ s negligible.

D. Update Rate

While a fast run time is desirable since it reduces the latency of the algorithms, the update rate is also of great interest. In other words, we not only want to have the position estimates as soon as possible, but we also want them as frequently as possible.

To measure the update rates, we divided the number of position updates by the duration of the trajectory,

$$\text{update rate} = \frac{\# \text{position updates}}{\text{duration}}.$$

We measured the durations of the ball trajectories manually by visualizing the event stream in Metavision Studio⁴. We did this for four different trajectories and report the mean and standard deviation in Table I in the column “Update rate”.

As listed in Table I, the frame-based baseline has with 149Hz the lowest update rate, which is dictated by the frame rate of the frame-based cameras. The event-based median filter has the highest update rate, with an average of $1.38 \cdot 10^4 \frac{\text{updates}}{\text{s}}$, which comes from the fact that it also has the lowest run time. The event-based particle filter ($3.49 \cdot 10^3 \frac{\text{updates}}{\text{s}}$) and the proposed event-based perception pipeline ($4.14 \cdot 10^3 \frac{\text{updates}}{\text{s}}$) are quite close together, with the latter being superior.

This experiment shows the benefits of an event-based perception pipeline. The update rate is not restricted by the sensor but by the algorithm used, which processes the event data.

E. Trajectory Prediction

After having detected the table tennis ball with both event-based cameras, the ball’s 3D position can be triangulated, as sketched in Fig. 1. These 3D positions can then be used to estimate the ball’s trajectory. The more accurate the ball’s predicted trajectory is, the better the robot arm can hit back. In this experiment, we used the Extended Kalman Filter (EKF)-based approach presented in [3] for the trajectory prediction. Next to the position, this EKF implementation also estimates the ball’s velocity and spin.

⁴https://docs.prophesee.ai/stable/metavision_studio

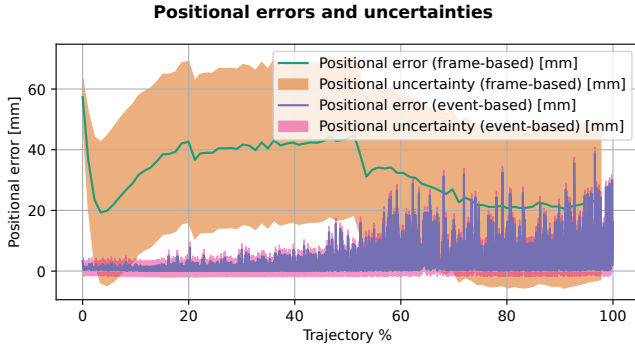


Fig. 7. The absolute positional error and the positional uncertainty of the trajectory with the proposed event-based perception pipeline and with the frame-based baseline.

More frequent 3D ball position updates can help the EKF to reduce its uncertainties of the position, velocity, and spin of the flying ball. As previous work discussed, the spin of the ball is crucial for a table tennis robot [3] [38] [39]. Therefore, having a lower uncertainty about the ball's spin is beneficial.

To show the benefits of our proposed event-based perception pipeline, we predicted the ball's trajectory with the mentioned EKF-based approach. We do this by providing the EKF the ball's 3D positions in the measurement update step and letting the EKF predict the next position, velocity, and spin. For a fair comparison, we estimated the transition covariance, the observation covariance, the initial state means, and the covariance of the initial state for both EKFs using the Expectation Maximization (EM) algorithm and made the predictions with these optimized EKF parameters. The EM algorithm optimizes the log-likelihood of the predicted states and, therefore, adjusts the parameters of the EKF to the data. This ensures good parameters for both EKF-based trajectory predictors (frame-based and event-based). Our implementation of EKF and the EM-based parameter optimization is based on `pykalman`⁵ which we adjusted to work with the non-linear model of the ball's trajectory mid-air, commonly used in the literature [40] [3].

We measure the error between the predicted ball position and the ball position from the triangulated trajectory. Since we do not have observations of the spin and velocity, we emphasize the uncertainties of the estimated velocities and spin values given by the EKF.

In Fig. 7, we visualized the positional errors and the corresponding uncertainties of our event-based pipeline and of the frame-based baseline. In Fig. 8, we show the uncertainties of the velocity and spin estimates of our event-based pipeline and of the frame-based baseline.

As we can see in Fig. 7, the positional error of the event-based pipeline drops faster at the start of the trajectory. We can also see that the positional error of the event-based pipeline is lower compared to the frame-based baseline most of the time. The difference in the course of the positional error between the frame-based and the event-based pipeline is partially influenced by the different camera setups. As

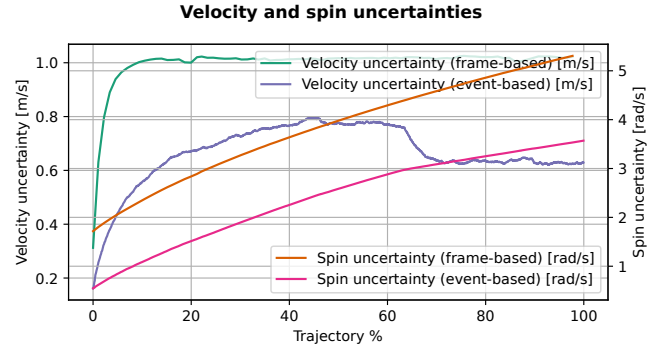


Fig. 8. The uncertainty of the predicted velocity and the uncertainty of the predicted spin of the trajectory with the proposed event-based perception pipeline and with the frame-based baseline.

sketched in Fig. 2, the two event-based cameras are located on the side of the robot, whereas the frame-based cameras are placed on all four corners of the ceiling.

Similar to the positional error, the uncertainties of the event-based pipeline are also smaller compared to the frame-based baseline, most of the time.

In Fig. 8, we can see that the uncertainties of both the velocity and the spin estimates are lower when using the event-based pipeline. This showcases the benefit of using an asynchronous event-based perception pipeline instead of a frame-based one.

V. CONCLUSION

Table tennis robots are a challenging research field for robotic perception algorithms and, therefore, have gained traction over the last few years. To the best of our knowledge, we present the first real-time event-based perception pipeline for a table tennis robot. While not yet integrated into a whole table tennis robot system, using a table tennis robot setup, we showcase the benefits of an event-based perception pipeline over perception pipelines that use traditional frame-based cameras. Namely, an update rate that is an order of magnitude faster than a frame-based perception pipeline while having a comparable accuracy. Furthermore, we show that the higher update rate results in lower uncertainties of the estimated position, velocity, and spin. This is beneficial for the path planning and robot control of a table tennis robot, which has to be fast to rally the ball back successfully.

We hope this work will motivate more researchers working on table tennis robots to use event-based computer vision.

ACKNOWLEDGMENTS

Special thanks to Sony AI for partially funding this project. We would also like to thank Mario Laux for his help with the C++ implementation of the ball detector. Thanks to Bernd Pfrommer for his open source contributions to many (event-based vision) ROS packages, two of them used in this work.

⁵<https://github.com/pykalman/pykalman>

REFERENCES

- [1] A. Ziegler, T. Gossard, K. Vetter, J. Tebbe, and A. Zell, “A multi-modal table tennis robot system,” in *RoboLectics: Workshop on Robot Learning in Athletics @CoRL 2023*, 2023. [Online]. Available: <https://arxiv.org/abs/2310.19062>
- [2] D. B. D’Ambrosio, S. Abeyruwan, L. Graesser, A. Iscen, H. B. Amor, A. Bewley, B. J. Reed, K. Reymann, L. Takayama, Y. Tassa, K. Choromanski, E. Coumans, D. Jain, N. Jaitly, N. Jaques, S. Kataoka, Y. Kuang, N. Lazic, R. Mahjourian, S. Moore, K. Oslund, A. Shankar, V. Sindhwani, V. Vanhoucke, G. Vesom, P. Xu, and P. R. Sanketi, “Achieving human level competitive robot table tennis,” 2024. [Online]. Available: <https://arxiv.org/abs/2408.03906>
- [3] J. Tebbe, Y. Gao, M. Sastre-Rienietz, and A. Zell, “A table tennis robot system using an industrial KUKA robot arm,” in *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 33–45. [Online]. Available: https://doi.org/10.1007/978-3-030-12939-2_3
- [4] D. D’Ambrosio, N. Jaitly, V. Sindhwani, K. Oslund, P. Xu, N. Lazic, A. Shankar, T. Ding, J. Abelian, E. Coumans, G. Kourretas, T. Nguyen, J. Boyd, A. Iscen, R. Mahjourian, V. Vanhoucke, A. Bewley, Y. Kuang, M. Ahn, D. Jain, S. Kataoka, O. Cortes, P. Sermanet, C. Lynch, P. Sanketi, K. Choromanski, W. Gao, J. Kangasputa, K. Reymann, G. Vesom, S. Moore, A. Singh, S. Abeyruwan, and L. Graesser, “Robotic table tennis: A case study into a high speed learning system,” in *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation, Jul. 2023. [Online]. Available: <https://doi.org/10.15607/rss.2023.xix.006>
- [5] S. Gomez-Gonzalez, Y. Nemmour, B. Schölkopf, and J. Peters, “Reliable real-time ball tracking for robot table tennis,” *Robotics*, vol. 8, no. 4, p. 90, Oct. 2019. [Online]. Available: <https://doi.org/10.3390/robotics8040090>
- [6] T. Ding, L. Graesser, S. Abeyruwan, D. B. D’Ambrosio, A. Shankar, P. Sermanet, P. R. Sanketi, and C. Lynch, “Learning high speed precision table tennis on a physical robot,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2022. [Online]. Available: <https://doi.org/10.1109/iros47612.2022.9982205>
- [7] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, “Event-based vision: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–180, 2022.
- [8] M. Monforte, A. Arriandaga, A. Glover, and C. Bartolozzi, “Exploiting event cameras for spatio-temporal prediction of fast-changing trajectories,” in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, Aug. 2020. [Online]. Available: <http://dx.doi.org/10.1109/AICAS48895.2020.9073855>
- [9] A. Mitrokhin, C. Fermuller, C. Parameshwara, and Y. Aloimonos, “Event-based moving object detection and tracking,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2018. [Online]. Available: <http://dx.doi.org/10.1109/IROS.2018.8593805>
- [10] B. Forrai, T. Miki, D. Gehrig, M. Hutter, and D. Scaramuzza, “Event-based agile object catching with a quadrupedal robot,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2023. [Online]. Available: <http://dx.doi.org/10.1109/ICRA48891.2023.10161392>
- [11] E. Perot, P. de Tournemire, D. Nitti, J. Masci, and A. Sironi, “Learning to detect objects with a 1 megapixel event camera,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 16 639–16 652. [Online]. Available: <https://proceedings.neurips.cc/paper/2020/file/c213877427b46fa96c639e837ccee-Paper.pdf>
- [12] T. Stoffregen, C. Scheerlinck, D. Scaramuzza, T. Drummond, N. Barnes, L. Kleeman, and R. Mahony, *Reducing the Sim-to-Real Gap for Event Cameras*. Springer International Publishing, 2020, p. 534–549. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-58583-9_32
- [13] A. Glover, A. Dinale, L. D. S. Rosa, S. Bamford, and C. Bartolozzi, “Iuvharris: A practical corner detector for event-cameras,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 12, p. 10087–10098, Dec. 2022. [Online]. Available: <http://dx.doi.org/10.1109/TPAMI.2021.3135635>
- [14] A. Glover, L. Gava, Z. Li, and C. Bartolozzi, “Edopt: Event-camera 6-dof dynamic object pose tracking,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2024, p. 18200–18206. [Online]. Available: <http://dx.doi.org/10.1109/ICRA57147.2024.10611511>
- [15] J. Billingsley, “Robot ping pong,” *Practical Computing*, vol. 6, no. 5, 1983.
- [16] D. Buchler, S. Guist, R. Calandra, V. Berenz, B. Scholkopf, and J. Peters, “Learning to play table tennis from scratch using muscular robots,” *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3850–3860, Dec. 2022. [Online]. Available: <https://doi.org/10.1109/tro.2022.3176207>
- [17] Y. Ji, X. Hu, Y. Chen, Y. Mao, G. Wang, Q. Li, and J. Zhang, “Model-based trajectory prediction and hitting velocity control for a new table tennis robot,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sep. 2021. [Online]. Available: <http://dx.doi.org/10.1109/IROS51168.2021.9636000>
- [18] L. Yang, H. Zhang, X. Zhu, and X. Sheng, “Ball motion control in the table tennis robot system using time-series deep reinforcement learning,” *IEEE Access*, vol. 9, p. 99816–99827, 2021. [Online]. Available: <http://dx.doi.org/10.1109/ACCESS.2021.3093340>
- [19] W. Gao, L. Graesser, K. Choromanski, X. Song, N. Lazic, P. Sanketi, Y. Sindhwani, and N. Jaitly, “Robotic table tennis with model-free reinforcement learning,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2020. [Online]. Available: <https://doi.org/10.1109/iros45743.2020.9341191>
- [20] Y. Gao, J. Tebbe, and A. Zell, “A model-free approach to stroke learning for robotic table tennis,” in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Jul. 2022. [Online]. Available: <https://doi.org/10.1109/ijcnn55064.2022.9892776>
- [21] J. Achterhold, P. Tobuschat, H. Ma, D. Büchler, M. Muehlebach, and J. Stueckler, “Black-box vs. gray-box: A case study on learning table tennis ball trajectory prediction with spin and impacts,” in *Proceedings of the Learning for Dynamics and Control Conference (L4DC)*, 2023.
- [22] M. Cannici, M. Ciccone, A. Romanoni, and M. Matteucci, “Asynchronous convolutional networks for object detection in neuromorphic cameras,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, Jun. 2019. [Online]. Available: <https://doi.org/10.1109/cvprw.2019.00209>
- [23] S. Schaefer, D. Gehrig, and D. Scaramuzza, “AEGNN: Asynchronous event-based graph neural networks,” in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2022. [Online]. Available: <https://doi.org/10.1109/cvpr52688.2022.01205>
- [24] Y. Liu, M. Gehrig, N. Messikommer, M. Cannici, and D. Scaramuzza, “Revisiting token pruning for object detection and instance segmentation,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024.
- [25] A. Glover and C. Bartolozzi, “Event-driven ball detection and gaze fixation in clutter,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2016. [Online]. Available: <https://doi.org/10.1109/iros.2016.7759345>
- [26] —, “Robust visual tracking with a freely-moving event camera,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sep. 2017. [Online]. Available: <http://dx.doi.org/10.1109/iros.2017.8206226>
- [27] D. Falanga, K. Kleber, and D. Scaramuzza, “Dynamic obstacle avoidance for quadrotors with event cameras,” *Science Robotics*, vol. 5, no. 40, Mar. 2020. [Online]. Available: <http://dx.doi.org/10.1126/scirobotics.aaz9712>
- [28] C. Scheerlinck, N. Barnes, and R. Mahony, *Continuous-Time Intensity Estimation Using Event Cameras*. Springer International Publishing, 2019, p. 308–324. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-20873-8_20
- [29] S. Seifozzakerini, W.-Y. Yau, B. Zhao, and K. Mao, “Event-based hough transform in a spiking neural network for multiple line detection and tracking using a dynamic vision sensor,” in *Proceedings of the British Machine Vision Conference 2016*, ser. BMVC 2016. British Machine Vision Association, 2016, pp. 94.1–94.12. [Online]. Available: <http://dx.doi.org/10.5244/C.30.94>
- [30] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, “Hots: A hierarchy of event-based time-surfaces for pattern recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 7, p. 1346–1359, Jul. 2017. [Online]. Available: <http://dx.doi.org/10.1109/TPAMI.2016.2574707>
- [31] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, “Hats: Histograms of averaged time surfaces for robust event-based object classification,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Jun. 2018. [Online]. Available: <http://dx.doi.org/10.1109/CVPR.2018.00186>
- [32] D. Gehrig, A. Loquercio, K. Derpanis, and D. Scaramuzza, “End-to-end learning of representations for asynchronous event-based data,” in *2019 IEEE/CVF International Conference on Computer Vision*

- (ICCV). IEEE, Oct. 2019, p. 5632–5642. [Online]. Available: <http://dx.doi.org/10.1109/ICCV.2019.00573>
- [33] T. Gossard, A. Ziegler, L. Kolmar, J. Tebbe, and A. Zell, “ewand: A calibration framework for wide baseline frame-based and event-based camera systems,” in *2024 International Conference on Robotics and Automation (ICRA)*. IEEE, 2024. [Online]. Available: <https://arxiv.org/pdf/2309.12685.pdf>
- [34] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>
- [35] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.
- [36] T. Delbruck and M. Lang, “Robotic goalie with 3 ms reaction time at 4% cpu load using event-based dynamic vision sensor,” *Frontiers in Neuroscience*, vol. 7, 2013. [Online]. Available: <http://dx.doi.org/10.3389/fnins.2013.00223>
- [37] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, “High speed and high dynamic range video with an event camera,” *IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI)*, 2019. [Online]. Available: http://rpg.ifi.uzh.ch/docs/TPAMI19_Rebecq.pdf
- [38] T. Gossard, J. Tebbe, A. Ziegler, and A. Zell, “Spindoe: A ball spin estimation method for table tennis robot,” in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2023. [Online]. Available: <http://dx.doi.org/10.1109/IROS55552.2023.10342178>
- [39] T. Gossard, J. Krismer, A. Ziegler, J. Tebbe, and A. Zell, “Table tennis ball spin estimation with an event camera,” in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, Jun. 2024, p. 3347–3356. [Online]. Available: <http://dx.doi.org/10.1109/CVPRW63382.2024.00339>
- [40] Z. Zhang, D. Xu, and M. Tan, “Visual measurement and prediction of ball trajectory for table tennis robot,” *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 12, p. 3195–3205, Dec. 2010. [Online]. Available: <http://dx.doi.org/10.1109/TIM.2010.2047128>

Bibliography

- Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., Kusnitz, J., Debole, M., Esser, S., Delbruck, T., Flickner, M., and Modha, D. (2017). A low power, fully event-based gesture recognition system. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.
- D'Ambrosio, D., Jaitly, N., Sindhwani, V., Oslund, K., Xu, P., Lazic, N., Shankar, A., Ding, T., Abelian, J., Coumans, E., Kouretas, G., Nguyen, T., Boyd, J., Iscen, A., Mahjourian, R., Vanhoucke, V., Bewley, A., Kuang, Y., Ahn, M., Jain, D., Kataoka, S., Cortes, O., Sermanet, P., Lynch, C., Sanketi, P., Choromanski, K., Gao, W., Kangaspunta, J., Reymann, K., Vesom, G., Moore, S., Singh, A., Abeyruwan, S., and Graesser, L. (2023). Robotic table tennis: A case study into a high speed learning system. In *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation.
- D'Ambrosio, D. B., Abeyruwan, S., Graesser, L., Iscen, A., Amor, H. B., Bewley, A., Reed, B. J., Reymann, K., Takayama, L., Tassa, Y., Choromanski, K., Coumans, E., Jain, D., Jaitly, N., Jaques, N., Kataoka, S., Kuang, Y., Lazic, N., Mahjourian, R., Moore, S., Oslund, K., Shankar, A., Sindhwani, V., Vanhoucke, V., Vesom, G., Xu, P., and Sanketi, P. R. (2024). Achieving human level competitive robot table tennis.
- Delbruck, T., Graca, R., and Paluch, M. (2021). Feedback control of event cameras. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE.
- Ding, T., Graesser, L., Abeyruwan, S., D'Ambrosio, D. B., Shankar, A.,

- Sermanet, P., Sanketi, P. R., and Lynch, C. (2022). Learning high speed precision table tennis on a physical robot. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- Dominguez-Morales, M. J., Jimenez-Fernandez, A., Jimenez-Moreno, G., Conde, C., Cabello, E., and Linares-Barranco, A. (2019). Bio-inspired stereo vision calibration for dynamic vision sensors. *IEEE Access*, **7**, 138415–138425.
- Falanga, D., Kleber, K., and Scaramuzza, D. (2020). Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, **5**(40).
- Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In *Image Analysis*, pages 363–370. Springer Berlin Heidelberg.
- Forrai, B., Miki, T., Gehrig, D., Hutter, M., and Scaramuzza, D. (2023). Event-based agile object catching with a quadrupedal robot. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- Gallego, G., Gehrig, M., and Scaramuzza, D. (2019). Focus is all you need: Loss functions for event-based vision. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.
- Gallego, G., Delbrück, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., Leutenegger, S., Davison, A. J., Conradt, J., Daniilidis, K., and Scaramuzza, D. (2022). Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **44**(1), 154–180.
- Gehrig, D., Gehrig, M., Hidalgo-Carrio, J., and Scaramuzza, D. (2020). Video to events: Recycling video datasets for event cameras. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.
- Gehrig, M., Aarents, W., Gehrig, D., and Scaramuzza, D. (2021). DSEC: A stereo event camera dataset for driving scenarios. *IEEE Robotics and Automation Letters*, **6**(3), 4947–4954.

- Glover, A. and Bartolozzi, C. (2016). Event-driven ball detection and gaze fixation in clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- Glover, A. and Bartolozzi, C. (2017). Robust visual tracking with a freely-moving event camera. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- Glover, A., Dinale, A., Rosa, L. D. S., Bamford, S., and Bartolozzi, C. (2022). luvharris: A practical corner detector for event-cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **44**(12), 10087–10098.
- Glover, A., Gava, L., Li, Z., and Bartolozzi, C. (2024). Edopt: Event-camera 6-dof dynamic object pose tracking. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, page 18200–18206. IEEE.
- Gomez-Gonzalez, S., Nemmour, Y., Schölkopf, B., and Peters, J. (2019). Reliable real-time ball tracking for robot table tennis. *Robotics*, **8**(4), 90.
- Gossard, T., Tebbe, J., Ziegler, A., and Zell, A. (2023). Spindoe: A ball spin estimation method for table tennis robot. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- Gossard, T., Ziegler, A., Kolmar, L., Tebbe, J., and Zell, A. (2024a). ewand: A calibration framework for wide baseline frame-based and event-based camera systems. In *2024 International Conference on Robotics and Automation (ICRA)*. IEEE.
- Gossard, T., Krismer, J., Ziegler, A., Tebbe, J., and Zell, A. (2024b). Table tennis ball spin estimation with an event camera. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, page 3347–3356. IEEE.

- Gossard, T., Schmalzl, J., Ziegler, A., and Zell, A. (2025a). Spin detection using racket bounce sounds in table tennis. In *2025 IEEE International Workshop on Sport, Technology and Research (STAR)*. IEEE.
- Gossard, T., Ziegler, A., , and Zell, A. (2025b). Tt3d: Table tennis 3d reconstruction. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE.
- Graça, R., McReynolds, B., and Delbruck, T. (2023). Shining light on the dvs pixel: A tutorial and discussion about biasing and optimization. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, page 4045–4053. IEEE.
- Hu, Y., Liu, S. C., and Delbruck, T. (2021). v2e: From video frames to realistic DVS events. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE.
- Huang, K., Wang, Y., and Kneip, L. (2021). Dynamic event camera calibration. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7021–7028. IEEE.
- Ji, Y., Hu, X., Chen, Y., Mao, Y., Wang, G., Li, Q., and Zhang, J. (2021). Model-based trajectory prediction and hitting velocity control for a new table tennis robot. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- Jiang, H., Sun, D., Jampani, V., Yang, M.-H., Learned-Miller, E., and Kautz, J. (2018). Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, page 9000–9008. IEEE.
- Joubert, D., Marcireau, A., Ralph, N., Jolley, A., van Schaik, A., and Cohen, G. (2021). Event camera simulator improvements via characterized parameters. *Frontiers in Neuroscience*, **15**.
- Kaiser, J., Tieck, J. C. V., Hubschneider, C., Wolf, P., Weber, M., Hoff, M., Friedrich, A., Wojtasik, K., Roennau, A., Kohlhaas, R., Dillmann, R.,

- and Zollner, J. M. (2016). Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks. In *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. IEEE.
- Kim, H., Leutenegger, S., and Davison, A. J. (2016). *Real-Time 3D Reconstruction and 6-DoF Tracking with an Event Camera*, page 349–364. Springer International Publishing.
- Kroeger, T., Timofte, R., Dai, D., and Gool, L. V. (2016). Fast optical flow using dense inverse search. In *Computer Vision – ECCV 2016*, pages 471–488. Springer International Publishing.
- Li, W., Saeedi, S., McCormac, J., Clark, R., Tzoumanikas, D., Ye, Q., Huang, Y., Tang, R., and Leutenegger, S. (2018). Interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset. In *British Machine Vision Conference (BMVC)*.
- Liu, J., Fang, Z., Zhang, K., and Tan, M. (2014). Improved high-speed vision system for table tennis robot. In *2014 IEEE International Conference on Mechatronics and Automation*. IEEE.
- Liu, Y., Gehrig, M., Messikommer, N., Cannici, M., and Scaramuzza, D. (2024). Revisiting token pruning for object detection and instance segmentation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*.
- Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision (darpa).
- Mitchelson, J. and Hilton, A. (2003). Wand-based multiple camera studio calibration. *Center Vision, Speech and Signal Process*.
- Mueggler, E. and Gehrig, D. (2014). Calibration toolbox by rpg group. https://github.com/uzh-rpg/rpg_dvs_ros/tree/master/dvs_calibration.

- Mueggler, E., Huber, B., and Scaramuzza, D. (2014). Event-based, 6-DOF pose tracking for high-speed maneuvers. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE.
- Mueggler, E., Rebecq, H., Gallego, G., Delbruck, T., and Scaramuzza, D. (2017). The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM. *The International Journal of Robotics Research*, **36**(2), 142–149.
- Muglikar, M., Gehrig, M., Gehrig, D., and Scaramuzza, D. (2021). How to calibrate your event camera. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE.
- Nair, G. B., Milford, M., and Fischer, T. (2024). Enhancing visual place recognition via fast and slow adaptive biasing in event cameras. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 3356–3363. IEEE.
- Nakabayashi, T., Higa, K., Yamaguchi, M., Fujiwara, R., and Saito, H. (2024). Event-based ball spin estimation in sports. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, page 3367–3375. IEEE.
- Nehvi, J., Golyanik, V., Mueller, F., Seidel, H.-P., Elgharib, M., and Theobalt, C. (2021). Differentiable event stream simulator for non-rigid 3d tracking. In *CVPR Workshop on Event-based Vision*.
- Perot, E., de Tournemire, P., Nitti, D., Masci, J., and Sironi, A. (2020). Learning to detect objects with a 1 megapixel event camera. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16639–16652. Curran Associates, Inc.
- Prophesee (2022). Calibration toolbox by prophesee. https://docs.prophesee.ai/metavision_sdk/modules/calibration/guides/intrinsics.html.

- Radomski, A., Georgiou, A., Debrunner, T., Li, C., Longinotti, L., Seo, M., Kwak, M., Shin, C.-W., Park, P. K. J., Ryu, H. E., and Eng, K. (2021). Enhanced frame and event-based simulator and event-based video interpolation network.
- Rebecq, H., Gehrig, D., and Scaramuzza, D. (2018). ESIM: an open event camera simulator. *Conf. on Robotics Learning (CoRL)*.
- Rebecq, H., Ranftl, R., Koltun, V., and Scaramuzza, D. (2019). High speed and high dynamic range video with an event camera. *IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI)*.
- Rehder, J., Nikolic, J., Schneider, T., Hinzmann, T., and Siegwart, R. (2016). Extending kalibr: Calibrating the extrinsics of multiple IMUs and of individual axes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- Rui Graca, Brian McReynolds, and Tobi Delbruck (2023). Optimal biasing and physical limits of dvs event noise.
- Schaefer, S., Gehrig, D., and Scaramuzza, D. (2022). AEGNN: Asynchronous event-based graph neural networks. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.
- Shin, K.-Y. and Mun, J. H. (2012). A multi-camera calibration method using a 3-axis frame and wand. *International Journal of Precision Engineering and Manufacturing*, **13**(2), 283–289.
- Stoffregen, T., Scheerlinck, C., Scaramuzza, D., Drummond, T., Barnes, N., Kleeman, L., and Mahony, R. (2020). *Reducing the Sim-to-Real Gap for Event Cameras*, page 534–549. Springer International Publishing.
- Tangram (2023). Tvcal: Deploy multi-sensor calibration. <https://www.tangramvision.com/sdk/multimodal-calibration>.
- Tebbe, J., Gao, Y., Sastre-Rienietz, M., and Zell, A. (2019). A table tennis robot system using an industrial KUKA robot arm. In *Lecture Notes in Computer Science*, pages 33–45. Springer International Publishing.

- Tebbe, J., Klamt, L., Gao, Y., and Zell, A. (2020). Spin detection in robotic table tennis. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- VLOGroup (2017). Calibration toolbox by vlo group. <https://github.com/VLOGroup/dvs-calibration>.
- Zhang, Z. (1999). Flexible camera calibration by viewing a plane from unknown orientations. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*. IEEE.
- Zhu, A. Z., Wang, Z., Khant, K., and Daniilidis, K. (2019). Eventgan: Leveraging large scale image datasets for event cameras.
- Ziegler, A., Gossard, T., Vetter, K., Tebbe, J., and Zell, A. (2023a). A multi-modal table tennis robot system. In *RoboLetics: Workshop on Robot Learning in Athletics @CoRL 2023*.
- Ziegler, A., Teigland, D., Tebbe, J., Gossard, T., and Zell, A. (2023b). Real-time event simulation with frame-based cameras. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- Ziegler, A., Joseph, D., Gossard, T., Moldovan, E., and Zell, A. (2025a). Biasbench: A reproducible benchmark for tuning the biases of event cameras. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE.
- Ziegler, A., Vetter, K., Gossard, T., Tebbe, J., Otte, S., and Zell, A. (2025b). Detection of fast-moving objects with neuromorphic hardware. In *2025 International Conference on Robotics and Automation (ICRA)*. IEEE.
- Ziegler, A., Gossard, T., Glover, A., and Zell, A. (2025c). An event-based perception pipeline for a table tennis robot. *IEEE Robotics and Automation Letters*. to be submitted.