University of Pennsylvania
ScholarlyCommons

Publicly Accessible Penn Dissertations

2019

Event-Based Algorithms For Geometric Computer Vision

Alex Zihao Zhu
University of Pennsylvania

Follow this and additional works at: https://repository.upenn.edu/edissertations

Part of the Artificial Intelligence and Robotics Commons, and the Robotics Commons

Recommended Citation
Zhu, Alex Zihao, "Event-Based Algorithms For Geometric Computer Vision" (2019). *Publicly Accessible Penn Dissertations*. 3566.
https://repository.upenn.edu/edissertations/3566

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/edissertations/3566
For more information, please contact repository@pobox.upenn.edu.

Event-Based Algorithms For Geometric Computer Vision

Abstract

Event cameras are novel bio-inspired sensors which mimic the function of the human retina. Rather than directly capturing intensities to form synchronous images as in traditional cameras, event cameras asynchronously detect changes in log image intensity. When such a change is detected at a given pixel, the change is immediately sent to the host computer, where each event consists of the x,y pixel position of the change, a timestamp, accurate to tens of microseconds, and a polarity, indicating whether the pixel got brighter or darker. These cameras provide a number of useful benefits over traditional cameras, including the ability to track extremely fast motions, high dynamic range, and low power consumption.

However, with a new sensing modality comes the need to develop novel algorithms. As these cameras do not capture photometric intensities, novel loss functions must be developed to replace the photoconsistency assumption which serves as the backbone of many classical computer vision algorithms. In addition, the relative novelty of these sensors means that there does not exist the wealth of data available for traditional images with which we can train learning based methods such as deep neural networks.

In this work, we address both of these issues with two foundational principles. First, we show that the motion blur induced when the events are projected into the 2D image plane can be used as a suitable substitute for the classical photometric loss function. Second, we develop self-supervised learning methods which allow us to train convolutional neural networks to estimate motion without any labeled training data. We apply these principles to solve classical perception problems such as feature tracking, visual inertial odometry, optical flow and stereo depth estimation, as well as recognition tasks such as object detection and human pose estimation. We show that these solutions are able to utilize the benefits of event cameras, allowing us to operate in fast moving scenes with challenging lighting which would be incredibly difficult for traditional cameras.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Kostas . Daniilidis

Keywords
event-based cameras, event cameras, self-supervised learning

Subject Categories
Artificial Intelligence and Robotics | Computer Sciences | Robotics

This dissertation is available at ScholarlyCommons: https://repository.upenn.edu/edissertations/3566

EVENT-BASED ALGORITHMS FOR GEOMETRIC COMPUTER VISION

Alex Zihao Zhu

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2019

Supervisor of Dissertation

Kostas Daniilidis, Professor, Computer and Information Science

Graduate Group Chairperson

Rajeev Alur, Professor, Computer and Information Science

Dissertation Committee

Camillo J. Taylor, Professor, Computer and Information Science

Jianbo Shi, Professor, Computer and Information Science

Daniel D. Lee, Professor, Electrical and Systems Engineering

Andrew Davison, Professor, Department of Computing, Imperial College London

EVENT-BASED ALGORITHMS FOR GEOMETRIC COMPUTER VISION

© COPYRIGHT

2019

Alex Zihao Zhu

This work is licensed under the

Creative Commons Attribution

NonCommercial-ShareAlike 3.0

License

To view a copy of this license, visit

`http://creativecommons.org/licenses/by-nc-sa/3.0/`

To my family and friends.

It would not have been possible without you.

ACKNOWLEDGEMENT

In hindsight, the last five years have gone by incredibly quickly. Coming into this program, I often felt out of my depth, and without the amazing support and friendship from everyone I've met along the way, I couldn't have possibly made it to this point.

Throughout my PhD, I've been extremely fortunate to be surrounded by a wonderful group of mentors to guide me. To my advisor, Kostas, who has been amazing in supporting me throughout the years, and guiding me through the difficult journey of being a PhD student, from helping me find a dissertation topic, even as I was struggling, to training me to becoming an expert in my field. You always have useful advice to give, regardless of the problem at hand, and it always amazes me how you seem to know everyone. To my committee, CJ Taylor, Jianbo Shi, Dan Lee and Andrew Davison, for helping me with feedback on my work and situating it in the field, as well as dealing with what was usually last minute scheduling. To Nikolay, who helped me publish my first set of papers, encouraged me to submit even when I had doubts, and helped me navigate my post-graduation options. I still use the techniques I learned from you in all of my papers. To Bernd, who helped me navigate the FLA program.

In addition to mentors, Penn is also an amazing source for colleagues. To all my coauthors, who I have had the pleasure of working with on a number of different projects. Research is rarely a single person's work, and this is definitely true in my case. To Stephen and George, who came into the program with me. Having people to share this experience has been invaluable, both in terms of friendship, as well as helping me whenever I was stuck on a problem. I wouldn't have been able to achieve half as much without you guys. To the Kostas group as a whole. The group has changed a lot throughout my years, but it hasn't diverged from being a close knit community whose support has been invaluable. To everyone in FLA, a project that lasted through a solid portion of my program, and kept research life interesting outside of submitting papers. I always looked forward to our trips, however stressful they may have been. To everyone in the GRASP lab, you have been a great help in both research and in intimidating other schools with our sheer numbers at conferences. I look forward to doing the same in the future.

Finally, I have to acknowledge my friends and family, who kept me sane throughout. To my dance family, PADT, and some of the closest friends I have made during my time at Penn. Special thanks to Gene for the introduction to this wonderful group. To the Robertson Scholars Program and Julian Robertson. If you had told me ten years ago that I would be completing my PhD in the US, I would not have believed you. This would not have been possible without your generosity and support. To my parents and my family, for supporting me throughout my life, and for providing a place to call home on the other side of the world. To my wonderful partner, Jimin, who is now sharing this experience with me. Thanks for keeping me sane, and for all the great times we've had together, with many more to come.

ABSTRACT

EVENT-BASED ALGORITHMS FOR GEOMETRIC COMPUTER VISION

Alex Zihao Zhu

Kostas Daniilidis

Event cameras are novel bio-inspired sensors which mimic the function of the human retina. Rather than directly capturing intensities to form synchronous images as in traditional cameras, event cameras asynchronously detect changes in log image intensity. When such a change is detected at a given pixel, the change is immediately sent to the host computer, where each event consists of the x,y pixel position of the change, a timestamp, accurate to tens of microseconds, and a polarity, indicating whether the pixel got brighter or darker. These cameras provide a number of useful benefits over traditional cameras, including the ability to track extremely fast motions, high dynamic range, and low power consumption.

However, with a new sensing modality comes the need to develop novel algorithms. As these cameras do not capture photometric intensities, novel loss functions must be developed to replace the photoconsistency assumption which serves as the backbone of many classical computer vision algorithms. In addition, the relative novelty of these sensors means that there does not exist the wealth of data available for traditional images with which we can train learning based methods such as deep neural networks.

In this work, we address both of these issues with two foundational principles. First, we show that the motion blur induced when the events are projected into the 2D image plane can be used as a suitable substitute for the classical photometric loss function. Second, we develop self-supervised learning methods which allow us to train convolutional neural networks to estimate motion without any labeled training data. We apply these principles to solve classical perception problems such as feature tracking, visual inertial odometry, optical flow and stereo depth estimation, as well as recognition tasks such as object detection and human pose estimation. We show that these solutions are able to utilize the benefits of event cameras, allowing us to operate in fast moving scenes with challenging lighting which would be incredibly difficult for traditional cameras.

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 The Human Visual System

Vision is an immensely important skill for a large proportion of animals, and particularly so for humans. In fact, around 25% of the human brain is specialized towards visual understanding [35]. Such a heavy allocation of resources towards this task underlines both its merit for our survival as well as the complexity involved.

Human visual understanding begins in our eyes[1], where light is focused onto a layer of cells on the retina, in the rear of the eye. This light passes through a set of photo-receptor cells, commonly known as rods and cones, whose membrane potentials alter with the amount of light arriving at each cell. These cells are connected to bipolar cells, which in turn relay information to ganglion cells. The ganglion cells have the role of firing action potentials down the optic nerve to the visual cortex, where high level visual understanding occurs. An interesting aspect of these ganglion cells is that they do not necessarily relay direct intensity information, such as the images we see in our minds and in photos. Instead, the firing rate from each ganglion cell correlates with changes in light intensity. That is, a ganglion cell's firing rate is maximized when the amount light arriving at its corresponding photo receptor cells changes, and minimized when it is static.

This change information, encoded as a firing rate, is sent to the primary visual cortex,

[1]This is not intended to be an in depth explanation of human anatomy. The interested reader is advised to refer to this excellent textbook [9].

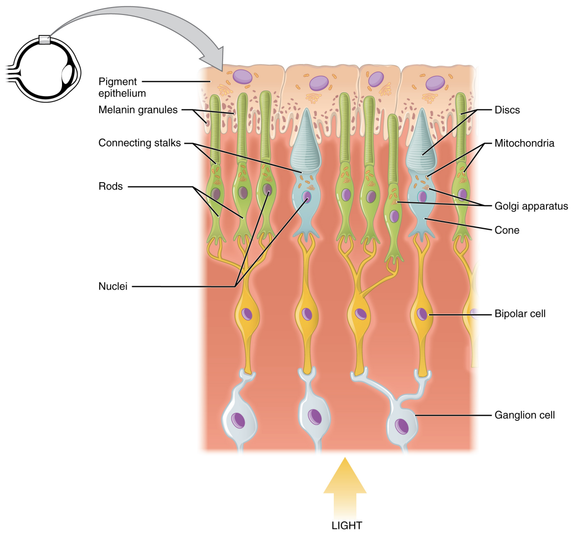


Figure 1: Anatomy of the human retina. Light passes through the transparent ganglion and bipolar cells, and trigger changes in the membrane potentials in the rod and cone cells. These signals are propagated through the bipolar cells to the ganglion cells, which fire action potentials to the visual cortex at a rate proportional to differences in light intensity arriving at each rod and cone. Illustration from Anatomy & Physiology, Connexions Web site. `http://cnx.org/content/col11496/1.6/`, Apr 4, 2019.

or V1, where low level understanding in the form of understanding oriented lines begins. These signals then propagate to the higher layers in the visual cortex, where complex understanding of higher order concepts such as shape, motion, and object recognition occur, and are eventually blended together to form the images we see in our minds.

While there is still a lot to be understood about the human visual system, this hierarchical model of the visual cortex forms the foundation behind the incredible set of perception tasks that humans are able to perform.

1.2 Traditional Cameras

Modern cameras mimic the output of the human visual system. The front-end optics operate with similar principles, by focusing light through a lens onto a sensor array.

However, as cameras were originally designed with a human audience in mind, they bypass most of the early layers of visual understanding in the brain. Instead, each pixel in the sensor array directly measures the number of photons arriving at that pixel, generating an image that resembles the final product we see in our minds. Due to the intrinsic benefits of parallelization for electronic devices, most cameras operate synchronously. That is, photons are integrated for exposure times which are fixed for every pixel on the sensor. Synchronizing the pixels allows for an entire image to be read out at once for global shutter cameras, and entire rows for rolling shutter cameras. This departs from the human visual model, where each ganglion cell operates independently, and fires asynchronously as light arrives at the retina.

From these artificial images, the field of computer vision has aimed to achieve human levels of understanding. Many classical methods involved reverse engineering the orientation selective V1 neurons by extracting gradients or edges from images [24, 32]. Indeed, modern deep learning pipelines have been shown to learn similar representations, with earlier layers extracting edge information, and later layers extracting progressively higher levels of understanding [161].

However, while it is certainly possible to work backwards from the final image in order to extract visual understanding, one might pose the question, is this the optimal starting point for an artificial visual understanding pipeline? In other words, can we design a camera which is more suited towards computer vision, where the goal is visual understanding, than the traditional ones designed for human entertainment? The human visual system certainly suggests that this is true, with many visual understanding tasks being performed without ever receiving a full image.

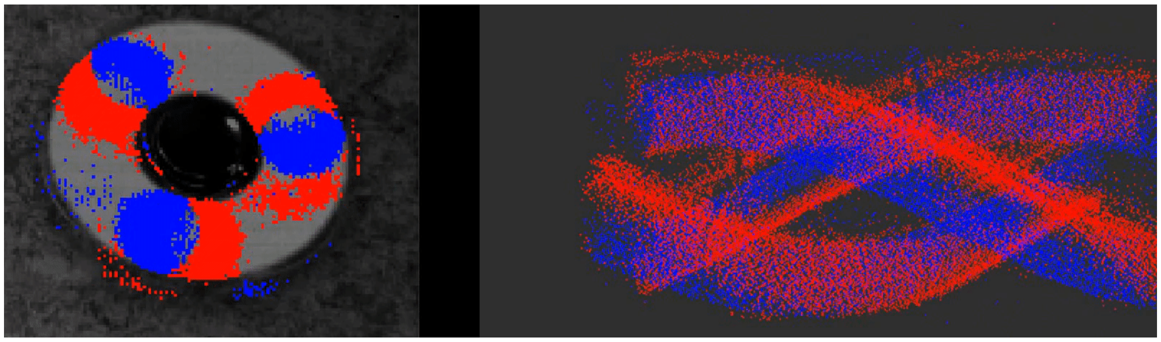


Figure 2: Events and grayscale image generated by a spinning fidget spinner recorded from a DAVIS 346b event camera[2]. Left: Grayscale image with events overlaid in 2D. Blue and red points indicate positive and negative events, respectively. Right: Visualization of the events along the t-y axes. Due to the high temporal resolution of the events, there is no motion blur or temporal aliasing in the 3D x-y-t space.

1.3 Event Cameras

Event cameras such as the DVS [84] and the ATIS [116] provide a possible positive response to this question. The front-end optics of these cameras are the same as traditional cameras, utilizing a lens to focus light onto a pixel array. However, rather than integrating photons over fixed exposure times, they aim to closely mimic the ganglion cells in the human visual system, which respond to changes in illumination. In particular, each pixel in the event camera asynchronously tracks changes in the log intensity at each pixel. When such a change is detected, the camera immediately sends an event to the host computer, consisting of the x, y pixel position of the change, a timestamp, t, which is accurate to tens of microseconds, and a binary polarity, p, which indicates whether the change was positive or negative. A sample of the output from the event camera can be found in Figure 2. By mimicking more closely the human visual system, one can hope that we can extract visual understanding more efficiently from event cameras than traditional ones.

[2]`https://inivation.com/dvs/`

Of course, biological plausibility should not be the only benefit of event cameras if we want to make the argument to use them over traditional cameras. Thankfully, these cameras exhibit a number of more direct benefits with many useful consequences for computer vision tasks.

1.4 Advantages of Event Cameras

1.4.1 High Speed Tracking

One such benefit stems directly from the asynchronous nature of the camera. As each event is sent immediately as it is detected, there is no longer a need to wait for a global exposure time. This allows processing with event cameras to react much faster than synchronous paradigms, otherwise known as ***low latency***. Combined with the high temporal resolution of each timestamp, these properties allow for ***tracking of incredibly fast motions***. In order to achieve similar tracking rates with a traditional camera, thousands of images need to be captured every second, generating an immense amount of data to be processed.

1.4.2 High Dynamic Range

Another common issue with traditional cameras is over or under exposure, where a correct exposure for one part of the image may be too high or low for another. This results in, at times, an irrecoverable loss of information when there are strong lighting variations within a scene. For robotics, these situations are relatively common, such as when driving into the sun, or transitioning from indoors to outdoors. As each pixel in an event camera is independent, they very rarely run into such issues. Formally, event cameras have much ***higher dynamic range*** than traditional cameras, with a dynamic range of around 140dB versus 60dB, respectively. In addition, as the camera tracks changes in log intensity, they also exhibit a consistent response over a very wide range of light intensities. That is, a much smaller change is required in the absolute magnitude for a darker scene to achieve the same log difference as a brighter scene. This gives the cameras ***excellent low light performance***, without the need to tune hyperparameters such as exposure time.

It is important to note that the low latency and high dynamic range properties are not independent when comparing to traditional cameras. For a synchronous camera with fixed global exposure time, the exposure must be increased as light intensity decreases in order to capture enough photons for an image. This results in increased latency for traditional images in dark scenes, and is the reason why images blur much more quickly in darker scenes. In these environments, event cameras become progressively more compelling, as even relatively slow motions may blur out a traditional image in low light conditions.

1.4.3 Low Power

Finally, event cameras have a significantly ***lower power consumption*** and (usually) ***lower bandwidth*** than traditional cameras. As only changes are streamed in the output, redundant information is not sent when parts of the image remain constant. This can have significant benefits for resource constrained scenarios such as mobile robotics or embedded devices.

In summary, the main direct benefits of event cameras are:

- Low latency and high temporal resolution.
- High dynamic range.
- Low power consumption.

1.5 Challenges with Event Cameras

Given these benefits, one may wonder why event cameras haven't been adopted more widely in the computer vision and robotics community. One major reason is simply practicality. Traditional cameras have been around for over 200 years, and today the number of cameras in the world sits in the tens of billions. Disrupting this technology with centuries of innovation with another vision based sensor is a challenging task, especially for a sensor that has been around for just over a decade.

However, there are also a number of technical challenges in trying to achieve the same level of success that computer vision has seen for traditional cameras. In particular, the three main challenges are the need to develop novel, asynchronous algorithms, the need for a replacement for the photometric loss, and the need to overcome the lack of training data for events.

1.5.1 Asynchronous Processing

Many of the major advantages of event cameras stem from their asynchronous operation. While many of these benefits, such as high dynamic range and low power, come as is, the latency benefits provided by the camera are only as good as the algorithm handling the event stream. If we pair the asynchronous output from the camera with a fully synchronous algorithm, we lose the ability to react to changes in the environment. If we under sample the event stream, we may not be able to react to fast changes in the scene, while if we over sample, we waste energy processing when not much has changed in the scene. The ideal solution to this would be a fully asynchronous algorithm, which has some internal state representing its belief of the scene, and updates this belief with every event as it arrives [1, 122, 80].

These asynchronous algorithms have seen success in extracting information from the events, promising extremely fast reaction times. However, most modern computing systems were designed with parallel computing in mind. Asynchronous processing, on the other hand, is an intrinsically sequential task, making efficient implementation of these algorithms without batching a significant challenge. In recent years, a new class of neuromorphic processors, such as the IBM TrueNorth [94] and the Intel Loihi [34], have emerged as a potential solution to this problem. These processors are by design asynchronous, and have the potential to allow for real time asynchronous processing directly from the camera. A number of works [56, 2, 3] have demonstrated real time algorithms running fully asynchronously on these processors.

However, these processors must encounter the same entrenched competitor problem that event cameras face with traditional cameras. Until neuromorphic computing becomes mainstream, we would ideally be able to develop algorithms which take advantage of the asynchronous nature of event cameras, while also utilizing the benefits provided by parallel computing. An alternative solution which satisfies both of these constraints is to process events in batches, but update the size of each batch asynchronously. This way, each batch can be processed in parallel, but the decision of when to collect the next batch is determined asynchronously, depending on the event stream. This introduces additional latency when waiting for the next batch of events over per event processing, but this is typically acceptable, as we rarely actually need microsecond level reactions. In order to asynchronously batch events, some kind of heuristic is needed to decide the size of the next batch. At this point, a large proportion of the event camera community has converged to the idea that having a constant amount of per-point displacement within the window is desirable. This heuristic is nice, as it provides some upper bound on the number of events generated within the window (although this is scene dependent), and the spatial distribution of the events is motion invariant. For tasks which aim to estimate the motion in the scene, this heuristic guarantees that updates will be provided along even intervals in the spatial domain. For example, if the goal is to estimate camera pose, it guarantees that there will be no large jumps in the true pose between updates.

One solution in this vein is the event lifetime [101]. This method uses the discretization of the pixel space to determine the time window required for a fixed displacement in the image. In particular, if we know the optical flow, $(\dot{x}, \dot{y})$ of a point, we should be able to reliably estimate the time needed for this point to travel one pixel, as this is simply:

$$\tau = \frac{1}{\|(\dot{x}, \dot{y})\|_2} \tag{1.1}$$

We can then set the time window within which to batch events to be $k \cdot \tau$, where k is the desired displacement within each window. However, this method is challenging to implement in practice for a number of reasons. The first is that estimating optical flow for events remains an expensive and challenging task. The other problem is the question of the scale at which to apply this windowing. Having a single temporal window size for the entire image assumes that the optical flow is constant within the image. This assumption is often violated in natural scenes, such as when there are strong depth variations. It is possible to split the image into sections and compute a separate lifetime for each, but the decision of how to perform this separation is challenging, with the optimal solution resulting in each pixel having its own lifetime, which is almost regressing to the fully asynchronous case. In Chapter 2, we utilize this method within a feature tracking pipeline to update our temporal window sizes. As our feature tracker computes the optical flow of each feature as a part of its pipeline, computing the lifetime for each feature comes at very little additional cost.

Another, simpler solution, is to assume that, for a given scene, the number of events generated is roughly proportional to the displacement within the image. In this case, a fixed number of events should correspond, once again roughly, to a fixed displacement in the image. This is a much more imprecise method than the event lifetimes, but does not require additional computation from the event stream, and is incredibly trivial to implement. Several works have adopted this scheme, such as [121, 104], and the works in Chapters 5 and 7.

1.5.2 Space-time Disentanglement

Another benefit of synchronous processing is the disentanglement between space and time. By having every pixel report its intensity value at the same time, each image gives us a snapshot in time, such that we can separate spatial reasoning, such as finding edges in an image, from temporal reasoning, such as finding the motion of an object. For asynchronous sensors, such as event cameras, the two dimensions are interwoven in the event stream. Given an event stream without any prior information, it is impossible to determine whether a new event corresponds to a new point in the image, or an older point that has already generated an event, moving to a new pixel location. This can be represented as a data association problem, where each new event must be associated with either a new point in the image, or an older event. Our work in Chapter 2, 3 proposes one way to resolve this problem, by using a deblurred window of events to represent the spatial distribution of points in the image.

1.5.3 Photometric Loss

One of the underlying foundational concepts for traditional computer vision has been the photoconsistency assumption [64]. That is, a point in the world should have the same intensity in the image, regardless of confounding factors such as viewpoint. This assumption allows us to apply a photometric loss to solve for the data associations between points in multiple images. For example, the correct optical flow should be the one which maps a patch in one image to the one most similar in intensity in the next. This assumption is pervasive in a large proportion of geometric vision tasks, from optical flow [5] to stereo matching [20] and direct visual odometry [36]. Having such a measure of correspondence is incredibly useful, as data association is one of the most challenging problems in geometric computer vision.

Unfortunately, events no longer directly contain any intensity information (although several works have shown that the grayscale image can be reconstructed for a given batch of events [6, 126]). As each event only represents a change in the image, changes between very low intensities and very high intensities will generate the same events (up to some noise).

The loss of a photometric loss is a significant impediment between porting traditional computer vision algorithms directly to events. In Section 1.6, we will propose a substitute for this loss for motion estimation, by utilizing the motion blur induced when projecting events to the 2D image plane.

1.5.4 Lack of Training Data

This is less an intrinsic property unique to the camera itself, but rather a common issue for any new sensor. As event cameras are less than a decade old, there simply are not many datasets with event data. For the current wave of deep learning in computer vision, good data is arguably one of the most important parts of developing a state of the art learning pipeline. While we wait for datasets to be developed, one way to take advantage of the benefits of deep learning in the meantime is through unsupervised methods. In particular, we focus on unsupervised motion estimation, by using known geometric relationships between the desired motion and image formation. In Chapter 6, we use a photometric loss applied to the grayscale images generated simultaneously on some event cameras such as the DAVIS [18] to estimate optical flow. We extend this in Chapter 7, where we replace the photometric loss with a motion blur loss applied directly to the events, and learn optical flow as well as egomotion and depth. Finally, in Chapter 8, we leverage the large amounts of training data available for images by generating synthetic events from images via a neural network.

1.6 Induced 'Motion Blur' for Events

One of the main selling points of the asynchronous nature of event cameras is that they do not suffer from motion blur. It seems, then, counterintuitive that we can use motion blur to help us extract information from the event stream. It should help, perhaps, to specify that the motion blur here is blur induced by projecting the events into the 2D image plane, as seen in Figure 3b. That is, we can artificially generate images with motion trails analogous to the motion blur seen in normal images, where edges are smeared along the direction of motion. While this has largely been a problem for traditional cameras, we can actually use this induced blur to inform our algorithms when performing motion estimation, as we still have accurate timestamp information for each point. To see why this is the case, let us first model the problem.

As a simple example, let us assume that we have an image of a white circle on a black background which is moving with constant optical flow $(\dot{x}, \dot{y})$. Let the points on the circle be $\{(\chi_j, \gamma_j)\}$. Over time, events, $\{x_j(t), y_j(t), t, p_j(t) | t \in T\}$ are generated by each point as it moves into a new pixel, which we can model given the optical flow, where we ignore the polarity for now:

$$(x_j(t), y_j(t)) = (\chi_j, \gamma_j) + (\dot{x}, \dot{y})t \tag{1.2}$$

If we were to generate an image with these events, we would expect to see the circle, plus 'motion blur' in the direction of the optical flow. Note that an event image in this document will refer to the image where each pixel is a count of the number of events at that pixel, unless otherwise specified.

Given a set of events, and their corresponding optical flow, we can then perform the inverse, and recover the original points on the circle:

$$(\hat{\chi}_j, \hat{\gamma}_j) = \frac{1}{N} \sum_{t \in T} (x_j(t), y_j(t)) - (\dot{x}, \dot{y})t \tag{1.3}$$

If we had the correspondences, j, between the points on the circle and the events, we could easily solve for the optical flow by solving the least squares problem:

$$\min_{(\dot{x}, \dot{y})} \| (\hat{\chi}_j, \hat{\gamma}_j) - (\chi_j, \gamma_j) \|_2^2 \tag{1.4}$$

Unfortunately, these correspondences are difficult to determine in practice, and most of the time, we do not have a clear model of $\{(\chi_j, \gamma_j)\}$. A visualization of the deblurring can be found in Figure 3.

One possible method for estimating these points is by leveraging the edge points in the corresponding grayscale image [140, 46]. This provides a high quality estimate of the template points, and has shown to allow for reliable tracking. However, these methods are limited to situations where grayscale images work, prohibiting their use in for example high speed or low light scenes.

In the more general case where we do not have any knowledge about the image points, can we still use this deblurring to obtain some information about the scene? Of course, the answer is yes (why ask otherwise?). While we may not know what the deblurred image should look like, we do know certain properties of deblurred and blurred images that should hold in general. Specifically, we would expect the deblurred event image to cluster many events together, while the blurry image should spread them apart. While this may sound like a very vague heuristic, we can formalize this in various ways to allow us to either explicitly or implicitly solve this data association problem.

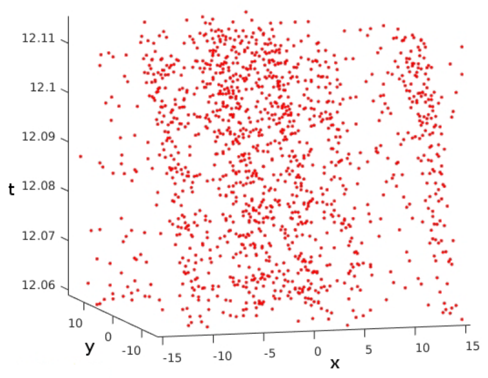
This concept has also been utilized in a number of other works outside of those presented in this thesis. Gallego et al. [43] proposed this idea of motion blur concurrently with our work in Chapter 2, where they use a measure of motion blur via the image variance as a cost function to estimate optical flow and angular velocity, respectively. This image variance loss is formalized in their later work [45], which provides a framework for using the loss in a general setting. Rebecq et al. [118] use the pose of a single camera from multiple views to generate a disparity space volume, in which the correct depth is similarly deblurred. Rebecq et al. [121] use a state estimator with pose and sparse depths to generate 'motion compensated' event images, on which they perform feature tracking. More recently, Mitrokhin et al. [96] use the synchronized images to perform object detection and tracking.

In Chapter 2, we propose a method which explicitly solves the data association through deblurring. Intuitively, the algorithm assumes that, given the correct deblurring, events generated by the same image point will be clustered together. We can thus estimate the data associations between events by modeling the probability of association as a Gaussian in the deblurred image.

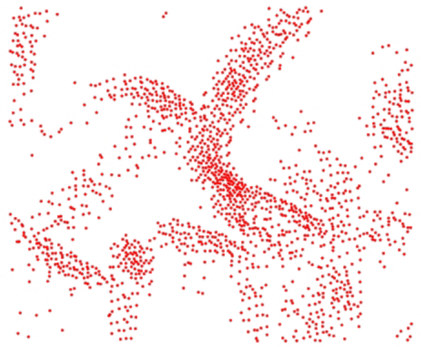
In Chapter 7, we utilize a method which instead implicitly solves the data association through deblurring. This is done by applying a loss directly on an image of the average timestamp at each pixel after deblurring. Here, the data association is implicitly assumed to be between events which arrive at the same pixel after deblurring.

Of course, we are not the only ones to use deblurring as a loss function for events. To the best of our knowledge, we published the first work with this method concurrently with Gallego et al. [43], who proposed maximizing image variance on the deblurred event image as an objective. This has been nicely formalized in their later work [45]. Mitrokhin et al. [96] have also proposed a loss which minimizes the average timestamp at each pixel, which we adopt in Chapter 7.

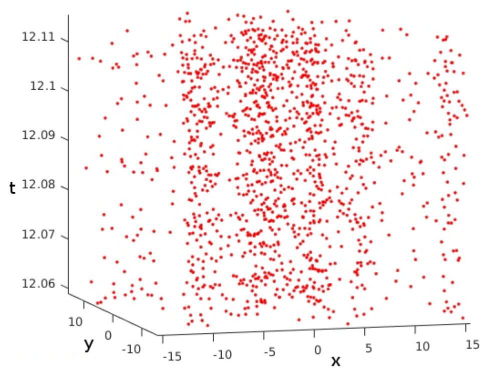
Overall, we can think of this deblurring loss as a substitute for the photometric loss used in traditional images, when estimating motion parameters such as optical flow or camera motion. In our work, we have utilized this property for optical flow and feature tracking, Chapter 2, stereo depth estimation, Chapter 5, and unsupervised learning, Chapter 7.



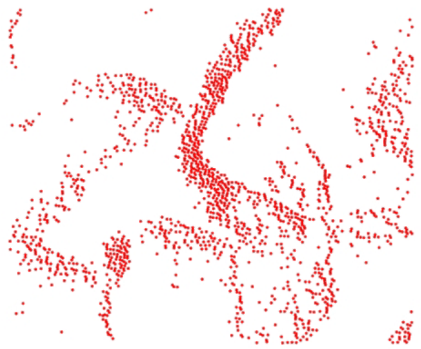
(a)



(b)



(c)



(d)

Figure 3: Visualization of events in a small spatiotemporal window (31pix.×31pix×0.5s), before and after deblurring. Recorded from cars driving by on the highway, with the object on the left in (d) being the rear of a car (the circle at the bottom is a wheel). (a) Raw events in 3D x-y-t space. No motion blur occurs in this space, and the optical flow can be seen to form straight lines moving to the left in the events. (b) Raw events projected to the x-y plane (i.e. by dropping the timestamps via orthogonal projection). Without the timestamps, motion blur occurs, generating a blurry event image. (c) Events with x, y positions deblurred according to (1.3). The previously sloped lines are now vertical, indicating zero optical flow within the deblurred events. (d) Projection of the deblurred events to the x-y plane. Lines now appear crisper, as events are clustered together.

Chapter 2

Event-based Feature Tracking with Probabilistic Data Association

2.1 Introduction

In this chapter, we propose a method for event-based feature tracking which uses the motion blur technique described in Section 1.6. In particular, we design an algorithm which, given a set of feature positions in the image plane, will estimate the positions of the features over time as they move through the image plane, using only the event stream. These feature tracks have a number of useful downstream applications, such as visual odometry, object tracking and 3D mapping. We take inspiration from Good Features to Track [135], and decompose each feature tracking step into an optical flow and affine alignment problem. Each problem is solved by addressing the data association for each event to a feature as a hidden soft random variable. The associations are regarded as probabilities because we do not need to make a hard commitment of an event to a feature. We apply an expectation-maximization (EM) scheme, where given optical flow we compute probabilities (weights) for data association and then we take the expectation over these probabilities in order to compute the optical flow. The computed optical flow is then used to deblur the events, forming a deblurred event image. This allows us to then compute an affine warping between the current set of events and a pre-computed initial template, allowing for drift-free tracking over time. Features are ultimately dropped based on the quality of

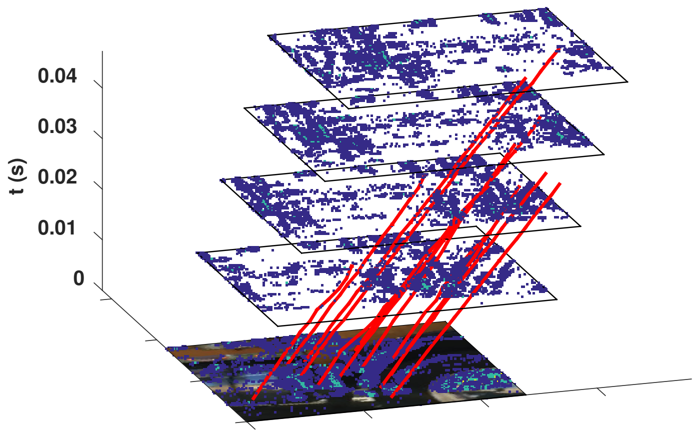


Figure 4: Selected features tracked on a truck driving at 60 miles/hr, 3 meters from the camera. Intermediate images are generated by integrating events for a period equal to three times their lifetimes.

the alignment as well as the convergence of the EM iteration. Grouping of the events into a feature is not by a fixed spatiotemporal window but rather by a lifetime [101] defined by a fixed length of the optical flow computed in the previous steps.

We show in egomotion as well as in very fast motion scenarios that we can track robustly features over long trajectories. In this chapter, we make the following novel contributions to the state of the art of event-based tracking:

- Events are grouped into features based on lifetime defined by the length of the optical flow.
- Assignment of events to existing features is soft and computed as probability based on a predicted flow.
- Flow is computed as a maximization of the expectation over all data associations.
- Deformation of the feature is modeled as affine and the residual of the affine fit serves as a termination criterion

2.2 Related Work

Feature tracking has been a topic that has attracted great interest from the research community. Due to the low latency and high temporal resolution of event cameras, we have the potential to track much faster motions than previously possible with standard cameras. Early work by Litzenberger et al. [87], inspired by mean-shift tracking [31], create clusters of events by assigning events to the closest centroid. Each cluster is weighted by the mean frequency of the events and inactive clusters are eliminated. Kim et al. [72] estimate a 3D-rotation for the purpose of mosaicking by updating a particle filter with the likelihood of each new event given the current pose estimate. Ni et al. [107] propose an approach where an event is assigned to the spatially closest feature model and its euclidean transformation and scaling with respect to the model is computed. Initialization is achieved by fitting spatiotemporal planes to the event space, as in Benosman et al. [13]. Lagorce et al. [79] define features using the Hough transform and then assign events using the ICP principle. Tschechne et al. [143] introduced the notion of a motion streak using biological vision principles where event tracks are detected by tuning spatiotemporal orientation over a longer temporal support. Alzugaray et al. [1] propose a novel local region descriptor for events, and represent the feature tracking problem as a graph search problem, where each tracked feature is a separate graph. Each new descriptor is then matched to the feature graph it is most likely associated with. Tedaldi et al. [140] and Kueng et al. [77] were the first to combine a frame-based camera and event-based sensor on the same pixel-array for tracking. Using a corner and an edge detector this approach initializes a feature patch which is enhanced by new events that are registered using a 2D euclidean transformation. The commitment of an event to a feature is hard and hence the registration is prone to false event associations. This work is extended by Gehrig et al. [46], who propose a probailistic model for event generation given a template generated from the images, and solve the data association problem with nonlinear optimization.

With the exception of [46], a common characteristic of the above approaches is the hard commitment, usually via ICP, to the assignment of an event to a model/feature with a subsequent estimate of a transformation given that commitment. Our approach integrates both data association and transformation estimation into a sound probabilistic scheme that makes it less prone to wrong correspondences. It does not make use of grayscale feature initializations. It is tested in very fast sequences where we show superiority over standard frame-based techniques.

2.3 Problem Formulation

2.3.1 Sensor Model and Event Propagation

Let $F \in \mathbb{R}^3$ and $f(t) \in \mathbb{R}^2$ be the projection of F onto the image plane at time t:

$$\begin{pmatrix} f(t) \\ 1 \end{pmatrix} \sim K \begin{bmatrix} R(t) & T(t) \end{bmatrix} \begin{pmatrix} F \\ 1 \end{pmatrix} \tag{2.1}$$

where K is the camera calibration matrix and $\begin{bmatrix} R(t) & T(t) \end{bmatrix}$ is the camera pose. In the remainder, we refer to the projections $f(t)$ as *features* and consider a set of features $\mathcal{F}(t)$. Given a feature $f \in \mathcal{F}(0)$, define a spatial window $B(s) := \{x \in \mathbb{R}^2 \mid \|x - f\| < s\}$. Let $\{P_j \in \mathbb{R}^3\}_{j=1}^m$ be a set of 3-D points, whose projections $\{p_j(0)\}_{j=1}^m$ onto the image plane at time 0 are contained within the window $B(s)$. Let $\mathcal{P}^f(t)$ denote the set of point projections associated with feature $f \in \mathcal{F}(0)$ at time t. At discrete times

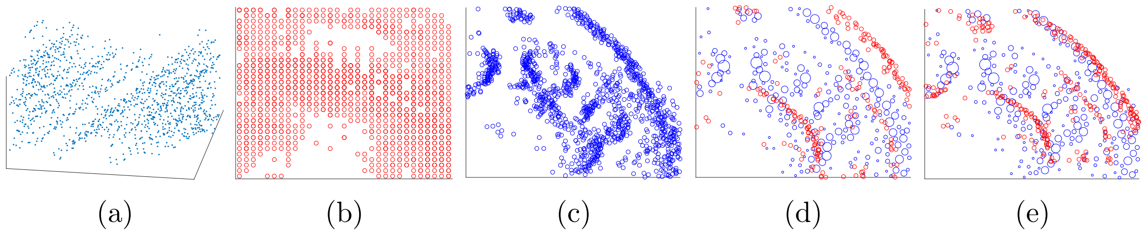


Figure 5: Graphical outline of the algorithm. (a) Event stream within the spatiotemporal window. Note the diagonal lines formed by the linear optical flow. (b) Events integrated directly onto the image with no flow correction. (c) Propagated events with estimated flow. Note the removal of the motion blur. (d) Later set of propagated events before affine warping. The size of the blue circles are the weights of each point after decimation. (e) Propagated events after affine warping.

$t_1, \dots, t_n$, the sensor generates a set of events $\{e_i := (x_i, t_i)\}_{i=1}^n$, where

$$x_i := p_{\pi(i)}(t_i) + \eta(t_i), \quad \eta(t_i) \sim \mathcal{N}(0, \Sigma), \quad \forall i$$

and $\pi : \{1, \dots, n\} \to \{1, \dots, m\}$ is an unknown many-to-one function representing the *data association* between the events $\{e_i\}$ and projections $\{p_j\}$ that generate them.

Problem (Event-based Feature Tracking)**.** *Given a set of events* $\mathcal{E}$ *generated by the point projections* $\bigcup_{t=0}^{T} \bigcup_{f \in \mathcal{F}(0)} \mathcal{P}^f(t)$*, estimate the feature projections* $\mathcal{F}(t)$ *in the image plane over time.*

2.4 Method

In Section 2.4.1, we introduce an optical flow based constraint within a spatiotemporal window. Section 2.4.2 then shows that we can optimize this constraint over the optical flow using the Expectation Maximization algorithm. The resulting flow can then be used to reconstruct the set of point projections within the spatial window, which we then use in Section 2.4.3 to refine the feature position using the EM-ICP algorithm [55]. Our tracking method then iterates between Section 2.4.2 and Section 2.4.3 to track a given set of features over time in the event stream. Section 2.4.4 outlines our technique to select the size of the temporal windows in an asynchronous fashion, Section 2.4.5 details our method for initializing the feature positions, and Section 2.4.6 summarizes our method for estimating the image features within each window. The entire algorithm is also summarized in Algorithm 1 and Figure 5.

2.4.1 Spatiotemporal Optical Flow Constraint

The motion of a feature $f(t) \in \mathcal{F}(t)$ in the image plane can be described using its *optical flow* $\dot{f}(t)$ as follows:

$$f(t) = f(0) + \int_0^t \dot{f}(s)ds = f(0) + tv(t), \tag{2.2}$$

where $v(t) := \frac{1}{t}\int_0^t \dot{f}(s)ds$ is the average flow of $f(0)$ over time. If t is sufficiently small, we can **assume** that the average flow v is constant and equal to the average flows of all point projections $\mathcal{P}(0)$ associated with $f(0)$. We can define a spatiotemporal window around $f(0)$ as the collection of events up to time t that propagate backwards onto $B(s)$:

$$W(s,t) := \{e_i \mid t_i < t, x_i - t_i v \in B(s)\} \tag{2.3}$$

Thus, provided that t is small, events corresponding to the same point in $\mathcal{P}(0)$ should propagate backwards onto the same image location. In other words, the following equality should hold for any pair $i, k \in [n]$ of events:

$$\|(x_i - t_i v) - (x_k - t_k v)\|^2 \mathbb{1}_{\{\pi(i)=\pi(k)=j\}} = 0, \quad \forall\, i,k \in [n] \tag{2.4}$$

However, since the data association π between events and 3D points is unknown, we can hope to satisfy the above requirement only in expectation:

$$\begin{aligned}\mathbb{E}_{\pi(i),\pi(k)}\|(x_i - t_i v) - (x_k - t_k v)\|^2 \mathbb{1}_{\{\pi(i)=\pi(k)=j\}} \\ = \left[\sum_{j=1}^{m} r_{ij} r_{kj}\right] \|(x_i - t_i v) - (x_k - t_k v)\|^2 = 0\end{aligned} \tag{2.5}$$

where $r_{ij} := \mathbb{P}(\pi(i) = j)$ and we assume that $\pi(i)$ is independent of $\pi(k)$.

Given an affine transformation (A, b) and the flow v of feature $f(0)$, we model the noise in the event generation process by defining the probability that event e_i was generated from point p_j as proportional to the pdf $\phi(A(x_i - t_i v) + b; p_j, \Sigma)$ of a Normal distribution with mean p_j and covariance Σ, i.e.,

$$r_{ij}(\{p_j\}) := \frac{\phi(A(x_i - t_i v) + b; p_j, \Sigma)}{\sum_{l=1}^{m} \phi(A(x_i - t_i v) + b; p_l, \Sigma)} \tag{2.6}$$

Where the argument $\{p_j\}$ is the set of points over which the means are defined. From here on, we will assume that r_{ij} with no argument implies that the set is the point projections $\{p_j\}$. Note also that Σ is a parameter to be experimentally tuned.

We propose an iterative approach to estimate the data association probabilities r_{ij} between the events $\{e_i^f\}$ and points $\{p_j^f\}$, the affine transformation A, b, and the optical flow v of feature f.

2.4.2 EM Optical Flow Estimation

In this section, we propose an Expectation Maximization algorithm for solving (2.5) over a spatiotemporal window $W(s, t)$ with a set of events $\{e_i, i \in [1, n]\}$. Within this window, our optical flow constraint becomes

$$\min_v \sum_{i=1}^{n} \sum_{k=1}^{n} \left[\sum_{j=1}^{m} r_{ij} r_{kj} \right] \| (x_i - t_i v) - (x_k - t_k v) \|^2 \tag{2.7}$$

In the E step, we update the r_{ij} and r_{kj}, given v using (2.6). Initially, the set of point positions $\{p_j\}$ is unknown, and so we first approximate the $\{p_j\}$ by the set of propagated events $\{x_i - t_i v\}$. In general, $x_i - t_i v \to p_{\pi(i)}$ as $v \to v'$, where v' is the true optical flow. In addition, as A and b are unknown, we initialize them as $A = I$ and $b = 0$. The full update, then, is $r_{ij}(\{e_i\})$.

The M step now involves solving for v given the r_{ij}. As we assumed that the average optical flow v is constant, (2.7) is a linear least squares problem in v, which corresponds to the general overdetermined system:

$$\begin{aligned} YD =& X \qquad (2.8) \\ \text{where } Y :=& v^T \\ D :=& [\sqrt{w_{12}}(t_1 - t_2), \dots, \sqrt{w_{1n}}(t_1 - t_n), \dots, \\ & \sqrt{w_{n(n-1)}}(t_n - t_{n-1})] \\ X :=& [\sqrt{w_{12}}(x_1 - x_2), \dots, \sqrt{w_{1n}}(x_1 - x_n), \dots, \\ & \sqrt{w_{n(n-1)}}(x_n - x_{n-1})] \\ w_{ik} :=& \sum_{j=1}^{n} r_{ij} r_{kj} \end{aligned}$$

To get the normal equations, we multiply both sides on the right by D^T:

$$Y = (XD^T)(DD^T)^{-1} = \frac{\sum_{i=1}^{n} \sum_{k=1}^{n} w_{ik}(x_i - x_k)(t_i - t_k)}{\sum_{i=1}^{n} \sum_{k=1}^{n} w_{ik}(t_i - t_k)^2} \tag{2.9}$$

We iterate equations (2.6) and (2.8) until convergence of the error (2.4). As in [55], we reject outlier matches by thresholding the likelihood w_{ik} when computing (2.8) by setting all the w_{ik} higher than some threshold ϵ to 0.

2.4.3 Feature Alignment

The flow estimate from Section 2.4.2 can then be used to propagate the events within the window to a common time t_0. Given the correct flow, this set of propagated events is then the approximation to the projection of the points P_j at time t_0, up to an affine transformation. As a result, given an estimate of the set of point projections at time t_0, $\{p_j := p_j(t_0) \in \mathbb{R}^2\}_{j=1}^{m}$, we can align the events with their corresponding points using a similar EM formulation as in Section 2.4.2. These estimated point projections are initialized on the first iteration, and the method is outlined in Section 2.4.6. The cost function for this alignment is that of the EM-ICP algorithm [55]:

$$\min_{A,b,r} \sum_{i=1}^{n} \sum_{j=1}^{m} r_{ij} \| A(x_i - t_i v) + b - p_j \|^2 \tag{2.10}$$

We can minimize this cost function using exactly the steps from Section 2.4.2. In the E step, we can use (2.6) to update r_{ij}.

The M step is also similar:

$$M : Y = (XD^T)(DD^T)^{-1} \tag{2.11}$$

where:

$$
\begin{aligned}
Y &:= \begin{bmatrix} A & b \end{bmatrix} \\
X &:= \begin{bmatrix} \sqrt{r_{11}}p_1, \dots, \sqrt{r_{1m}}p_m, \dots, \sqrt{r_{nm}}p_m \end{bmatrix} \\
D &:= \Bigg[\sqrt{r_{11}} \begin{pmatrix} x_1 - t_1 v \\ 1 \end{pmatrix}, \dots, \sqrt{r_{1m}} \begin{pmatrix} x_1 - t_1 v \\ 1 \end{pmatrix}, \dots, \\
&\quad \sqrt{r_{nm}} \begin{pmatrix} x_n - t_n v \\ 1 \end{pmatrix} \Bigg]
\end{aligned}
$$

As in Section 2.4.2, we iterate (2.6) and (2.11) until the error function (2.10) converges. We then use the new estimate for b to refine the prior estimate for the image feature positions, and propagate them to the next window as:

$$f_j(t_n) = f_j(t_0) - b + v(t_n - t_0) \tag{2.12}$$

Similarly, the point projections are propagated to the next time:

$$p_j(t_n) = p_j(t_0) + v(t_n - t_0) \tag{2.13}$$

If the EM fails to converge, or if the value of (2.10) is too high after convergence, we consider the tracker lost and abandon the feature.

2.4.4 Temporal Window Selection

Many event based techniques work with temporal windows of fixed size. However, these techniques have many similar drawbacks to traditional cameras, as the amount of information within the windows is highly variable depending on the optical flow within the image. Due to the quantization of the spatial domain, no information about the optical flow can be gained from the event stream until the projected points have moved at least one pixel within the image. On the other hand, too large of a window may violate the constant optical flow assumption made in Section 2.4.2. To ensure that the temporal windows are of an appropriate size, we dynamically adjust them using the concept of event 'lifetimes' [101]. Given the optical flow of a feature within a prior spatiotemporal window, we can estimate its lifetime τ as the expected time for the feature to move one pixel in the spatial domain:

$$\tau = \frac{1}{\|v\|} \tag{2.14}$$

For robustness against erroneous flow estimates, we estimate the lifetimes of several windows, and set the next temporal window size as k times the median lifetime. In our experiments, we observed that $k = 3$ was a reasonable value to avoid large optical flow deviations while still capturing a sufficient number of events. This technique can be extended to have separate temporal window sizes for each tracked feature for fully asynchronous tracking between features. However, we found that, in our testing, the variation in optical flow of our features was not large enough to require this.

2.4.5 Feature Selection

As this method relies on the assumption that the projected points are sparse, it will fail on spatial windows with dense points throughout. In addition, the matching scheme in Section 2.4.3 suffers from the same aperture problem as traditional feature matching techniques. To avoid selecting such windows for our tracker, we propagate all events within each temporal window onto the image plane with zero flow to generate an integrated image. As events are typically generated over edges in the image, this integrated image is similar to an edge map. We then use the Harris corner detector [57] to select spatial windows with edge orientations in multiple directions.

2.4.6 Point Set Generation

In order to perform the affine feature alignment step in Section 2.4.3, we must have an initial estimate of the set of point projections $\{p_j\}$. As the true point projections are unknown, we approximate them with the events in the first spatiotemporal window, propagated to the last time in the window, T, using the flow calculated in Section 2.4.2. This gives us a noisy set of points that approximate the true point projections, without motion blur. For each subsequent iteration, these points are propagated to the current time using (2.13), and aligned with the propagated events for that iteration. To reduce the computation time for matching against this potentially large feature set, we perform the sphere decimation algorithm in [55] to reduce the cardinality of this set.

2.5 Experiments

We present the results of our approach using a DAVIS-240C sensor [18] in two situations. First, we compare the tracking accuracy of our tracking algorithm on a structured, textured area at normal speeds against traditional image based tracking on the frame-based intensity values from the DAVIS. We then demonstrate qualitative results of our algorithm on tracking a vehicle on a highway traveling at roughly 60 miles/hr, which we qualitatively compare to the tracking results on the 240FPS output of an iPhone 6.

In each experiment, we used 31x31 pixel patches, with Σ_j set to $2 \times I_2$. At the beginning of each sequence, a manually picked integration time is selected to start the

Algorithm 1 Event-based Feature Tracking with Probabilistic Data Association

```
**Initialization**
    Initialize $\tau$ as $t'/k$ and integrate events for a short
      period of time over the image.
    Detect corner points using Harris corners on the
      integrated image, and initialize features $f_j$ at each
      corner.
**Tracking**
    Collect events for $k\tau$ seconds
    for each feature do
        $A \leftarrow I_2$, $b \leftarrow 0$, $v \leftarrow 0$, cost$\leftarrow \infty$, $\{p_j\} \leftarrow \{\}$
        while cost $> \epsilon$ do
            Find events within $W(s, k\tau)$ (2.3)
            Update $r_{ij}(\{p_j\})$ (2.6)
            Update $A$, $b$ (2.8)
            Calculate cost (2.7)
        end while
        Propagate events within the window to $t_0$ using $v$
        if $\{p_j\} = \{\}$ then
            $\{p_j\} \leftarrow$ propagated events
            continue
        end if
        cost$\leftarrow \infty$
        while cost $> \epsilon$ do
            Find events within $W(s, t)$ (2.3)
            Update $r_{ij}(\{p_i\})$ (2.6)
            Estimate $A$, $b$ and $\dot{x}$ using (2.11)
            Calculate cost (2.10)
        end while
        $\{p_j\} \leftarrow \{p_j\} - b + v \times k\tau$
    end for
    $\tau \leftarrow 1/\text{median}(\{\|v\|\})$
```

Figure 6: Images of a truck driving on a highway recorded from the 240 FPS video.

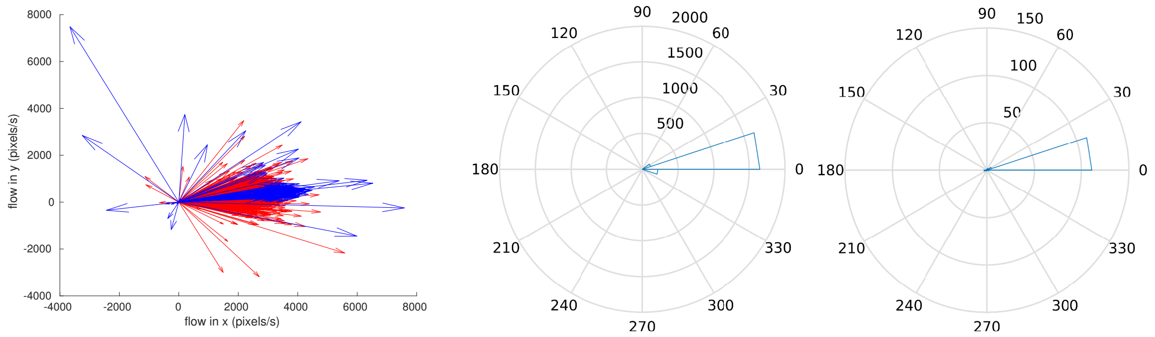


Figure 7: Left to right: (1) Optical flow estimates from our method (red) and KLT tracking (blue), (2) Polar histogram (20 bins) of optical flow directions estimated by our method, (3) Polar histogram (20 bins) of optical flow directions estimated by KLT.

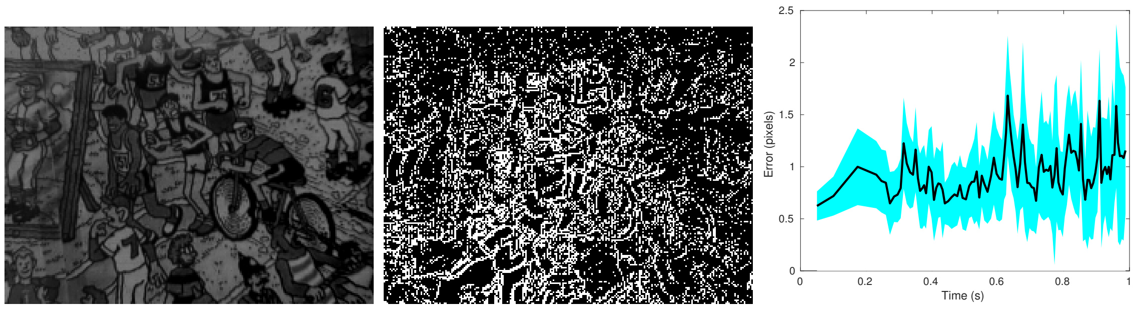


Figure 8: Left: Comparison between frame-based and integrated-event images. Right: Norm of feature position error between our method and KLT.

algorithm to guarantee that the first temporal window contained significant apparent motion in the spatial domain. However, this time is very robust, and we found that any integration time where points moved at least 2 pixels was sufficient. In both experiments, 20 features were generated, with new features initialized if fewer than 12 remained.

2.5.1 Comparison with Frame-Based Tracking

To quantitatively analyze the performance of our algorithm, we compare our results to the KLT Tracker [89] on a sequence where the DAVIS camera was moved in front of a textured surface (Figure 8). Due to the relatively low frame rate of 25Hz for the frame based images on the DAVIS, this motion was restricted to relatively low speeds. Features were initialized from the integrated event image, and tracked in both the event stream as well as the frame based images until the majority of features were lost in both trackers. During the one second tracking period, the features moved on average 100 pixels.

We show the mean tracking error for features that have not been discarded in Fig. 8, where the black line is the mean tracking error over all the features, and the cyan region is one standard deviation around the mean error. As the event based measurements arrive much faster than the frame based ones, we interpolate the event based feature position estimates to the nearest frame based position estimate in time using the event based optical flow estimate. The overall mean error from this technique is 0.9492 pixels, which is comparable to the state of the art in this topic [140].

2.5.2 Tracking on Scenes with High Apparent Motion

To test the algorithm on scenes with very high apparent motion, the camera was placed on the side of a highway with a speed limit of 60 miles per hour. Cars passed the camera at a distance between 3-4 meters, and passed the field of view in under 0.5s. We present here the results of tracking on a semi truck driving by at the posted speed limit. In this sequence, the average flow magnitude was 4000 pixels/s, and the (roughly) 15m long truck passed the camera's field of view in 800ms. The frame based images from the DAVIS sensor for these vehicles were almost completely blurred out. For comparison, we also recorded the scene with an iPhone 6 at 240 FPS (Figure 6), on which we also ran a KLT tracker. The 240 FPS video is sufficient to capture the motion in this sequence, but is beginning to show motion blur on the order of one or two pixels. The two cameras' extrinsic parameters were estimated using stereo calibration. Unfortunately, due to the relatively close distance of the vehicles to the camera, we were unable to accurately warp the images onto one another for a quantitative comparison, and so we will instead give qualitative comparisons for our flow estimation based on a warp of the iPhone images assuming that points all have a depth of 3 meters.

We visualize a subset of the feature tracks in Figure 4. It is interesting to note that, while the first integrated event image (superimposed over the iPhone image) has a significant amount of motion blur, the subsequent images have structures only a few pixels thick, due to the lifetime estimation from the optical flow.

In Fig 7, we analyze the distribution of the direction of the optical flow vectors estimated by our method and by the KLT tracker. We can see that the majority of flow vectors lie between 0 and 20 degrees. This can also be seen in the left-most plot in

Fig 7, which shows individual flow vectors, with optical flow calculated within tracks shorter than 20ms removed. From these plots, we can see that both the direction and magnitude of the KLT flow vectors are very similar, although they should not perfectly correspond.

2.6 Conclusions

In this chapter, we have presented a novel approach for feature tracking in asynchronous event-based sensors that relies on probabilistic data association. Estimating optical flow becomes, thus, not sensitive to erroneous associations of new events and is computed from the expectation over all associations. To increase persistence of our tracks we compute the affine transformation for each feature with respect to the starting time. Existing approaches use a hard correspondence commitment and usually compute a similitude transformation. The spatiotemporal support of our features is adaptive and defined by the size of the flow rather than a fixed time or a number of events. We show that it outperforms classic KLT trackers when they are applied to 240 FPS cameras capturing very fast motions of the order of one field of view per half a second. In Chapter 3, we extend this algorithm to allow us to track camera pose in a visual inertial odometry framework.

Chapter 3

Event-based Visual Inertial Odometry

3.1 Introduction

In this chapter, we refine the feature tracking method defined in Chapter 2, and wrap it in a visual inertial odometry pipeline. This pipeline allows us to estimate the 6dof pose of the camera from events and inertial measurements, without the need for any intensity frames. State of the art in visual-inertial odometry relies on feature-tracks over temporal windows [99]. Such tracks are difficult to obtain among asynchronous events due to the lack of any intensity neighborhood. We propose to use the previously defined data association scheme where multiple spatially neighboring events are softly associated with one feature whose motion is computed using the weighted event positions. Given the asynchronous feature tracks, filter estimates of the 3D rotation are used in a 2-feature RANSAC inlier selection with the translation direction as unknown. Given several feature tracks over time, we employ an Extended Kalman Filter which estimates all camera poses during the lifespan of the features. Similar to the MSCKF [99], we eliminate the depth from the measurement equations so that we do not have to keep triangulated features in the state vector.

Our contributions can be summarized as follows:

- A novel event association scheme resulting in robust feature tracks by employing two EM-steps and variable temporal frames depending on flow and rotation estimates obtained from the odometry filter.

- The first visual odometry system for event-based cameras that makes use of inertial information.
- We demonstrate results on very fast benchmark sequences and we show superiority with respect to classic temporally sparse KLT features in **high speed** and **high dynamic range** situations.

3.2 Related Work

Weikersdorfer et al. [149] present the first work on event-based pose estimation, using a particle filter based on a known 2D map. They later extend this work in [150] by fusing the previous work with a 2D mapping thread to perform SLAM in an artificially textured environment. Similarly, Censi et al. [27] use a known map of active markers to localize using a particle filter. Gallego et al. [44] also assume a known set of images, poses and depth maps, in order to perform 6dof pose estimation using an EKF. Several methods also combine an event-based camera with other sensors to perform tracking. The work by Censi et al. [26] combines an event-based camera with a separate CMOS camera to estimate inter-frame motions of the CMOS camera using the events in order to estimate camera velocity. Similarly, Tedaldi et al. [140] use the image frames from a DAVIS camera to perform feature detection and generate a template to track features in the event stream, which Kueng et al. [78] wrap in a standard visual odometry framework to perform up to scale pose estimation. In Weikersdorfer et al. [151], an event-based camera is combined with a depth sensor for 3D mapping, and uses the particle filter from Weikersdorfer et al. [149] for localization. However, fusing an event based camera with a CMOS or depth camera incurs the same costs as methods that use either camera, such as motion blur and limited dynamic range. An alternative approach for event-based pose tracking relies on jointly estimating the original intensity based image and pose. Kim et al. [73] pose the problem in an

EKF framework, but the method is limited to estimating rotation only. In [74], the authors extend this work and use the method by Bardow et al. [6] to jointly estimate the image gradient, 3D scene and 6dof pose. Our method is a 6dof tracking method that works without any prior knowledge of the scene.

In comparison to the latest work at the time in Kueng et al. [78], our method does not require any image frames, and our tracking algorithm treats data associations in a soft manner. By tracking features solely within the event stream, we are able to track very fast motions and in high dynamic range situations, without needing to reconstruct the underlying image gradient as in Kim et al. [74]. Our method also uses soft data associations, compared to [78] and [74], who make a hard decision to associate events with the closest projection of a 3D landmark, leading to the need for bootstrapping in [78]. By fusing the tracking with information from an IMU, we are also able to fully reconstruct the camera pose, including the scale factor, which vision only techniques are unable to do.

In more recent work, Mueggler et al. [104] cast the visual inertial odometry problem in a continuous space, by fitting a set of splines to the camera trajectory, which they solve assuming a known 3D map. Rebecq et al. [120] solve this problem in the more general case, by performing feature tracking in the event stream and fusing this information with an IMU using nonlinear optimization. The feature tracking is performed by deblurring the events using an estimate of the camera pose obtained from the IMU, and then using a traditional KLT tracker [89, 134] on the deblurred event images. This work is extended by Vidal et al. [144] who combine the event features with image features to improve robustness of the system.

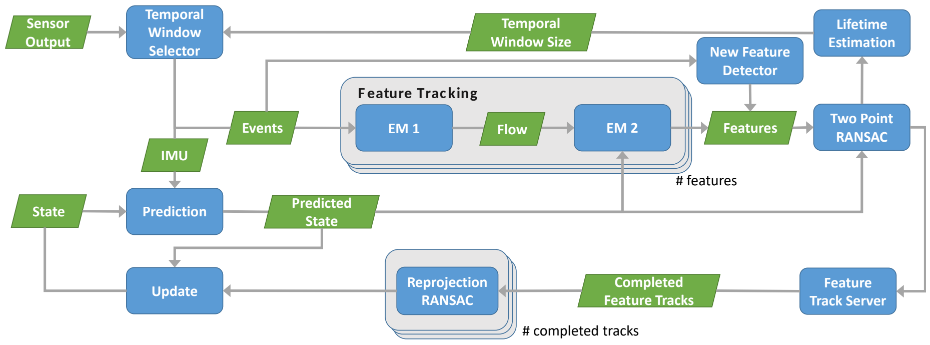


Figure 9: EVIO algorithm overview. Data from the event-based camera and IMU is processed in temporal windows determined by our algorithm. For each temporal window, the IMU values are used to propagate the state, and features are tracked using two expectation maximization steps that estimate the optical flow of the features and their alignment with respect to a template. Outlier correspondences are removed, and the results are stored in a feature track server. As features are lost, their feature tracks are parsed through a second RANSAC step, and the resulting tracks are used to update the sensor state. The estimated optical flows for all of the features are then used to determine the size of the next temporal window.

3.3 Problem Formulation

Consider a sensor package consisting of an inertial measurement unit (IMU) and an event-based camera. Without loss of generality, assume that the camera and IMU frames are coincident.[1] The state of the sensor package:

$$s := \begin{bmatrix} \bar{q} & b_g & v & b_a & p \end{bmatrix} \tag{3.1}$$

consists of its position $p \in \mathbb{R}^3$, its velocity $v \in \mathbb{R}^3$, the orientation of *the global frame in the sensor frame* represented by a unit quaternion $\bar{q} \in SO(3)$[2], and the accelerometer and gyroscope measurement biases, b_a and b_g, respectively.

At discrete times $\tau_1, \tau_2, \ldots$, the IMU provides acceleration and angular velocity mea-

[1]In practice, extrinsic camera/IMU calibration may be performed off-line [41]. The IMU and camera frames in the DAVIS-240C are aligned.

[2]The bar emphasizes that this quaternion is the conjugate of the unit quaternion representing the orientation of the sensor in the global frame.

surements $\mathcal{I} := \{(a_k, \omega_k, \tau_k)\}$. The environment, in which the sensor operates, is modeled as a collection of landmarks $\mathcal{L} := \{L_j \in \mathbb{R}^3\}_{j=1}^m$.

At discrete times $t_1, t_2, \ldots$, the event-based camera generates events $\mathcal{E} := \{(x_i, t_i)\}$ which measure the perspective projection[3] of the landmark positions as follows:

$$\begin{aligned}
\pi\left([X\;Y\;Z]^T\right) &:= \frac{1}{Z}\begin{bmatrix} X \\ Y \end{bmatrix} \\
h(L, s) &:= \pi\left(R\left(\bar{q}\right)(L - p)\right) \\
x_i &= h(L_{\alpha(i)}, s(t_i)) + \eta(t_i), \quad \eta(t_i) \sim \mathcal{N}(0, \Sigma)
\end{aligned} \tag{3.2}$$

where $\alpha : \mathbb{N} \to \{1, \ldots, m\}$ is an unknown function representing the data association between the events $\mathcal{E}$ and the landmarks $\mathcal{L}$ and $R(q)$ is the rotation matrix corresponding to the rotation generated by quaternion q.

Problem 1 (Event-based Visual Inertial Odometry)**.** *Given inertial measurements $\mathcal{I}$ and event measurements $\mathcal{E}$, estimate the sensor state $s(t)$ over time.*

3.4 Overview

The visual tracker uses the sensor state and event information to track the projections of sets of landmarks, collectively called features, within the image plane over time, while the filtering algorithm fuses these visual feature tracks with the IMU data to update the sensor state. In order to fully utilize the asynchronous nature of event-based cameras, the temporal window at each step adaptively changes according to the optical flow. The full outline of our algorithm can be found in Figure 9 and Algorithm 2.

[3]Without loss of generality, the image measurements are expressed in normalized pixel coordinates. Intrinsic calibration is needed in practice.

Our tracking algorithm leverages the property that all events generated by the same landmark lie on a curve in the spatiotemporal domain and, given the parameters of the curve, can be propagated along the curve in time to a single point in space (barring error from noise). In addition, the gradient along this curve at any point in time represents the optical flow of the landmark projection, at that time. Therefore, we can reduce the tracking problem to one of finding the data association between events and landmark projections, and estimating the gradient along events generated by the same point.

To simplify the problem, we make the assumption that optical flow within small spatiotemporal windows is constant. That is, the curves within these windows become lines, and estimating the flow is equivalent to estimating the slope of these lines. To impose this constraint, we dynamically update the size of the temporal window, dt, based on the time for each projected landmark position to travel k pixels in the image, estimated using the computed optical flow. This way, we are assuming constant optical flow for small displacements, which has been shown to hold in practice. Given these temporal windows, we compute feature position information within a discrete set of non-overlapping time windows $\{[T_1, T_2], [T_2, T_3], \dots\}$ where $T_{i+1} - T_i = dt_i$.

Problem 1a (Event-based Feature Tracking)**.** *Given event measurements $\mathcal{E}$, the camera state $s_i := s(T_i)$ at time T_i and a temporal window size dt_i, estimate the feature projections $\mathcal{F}(t)$ for $t \in [T_i, T_{i+1}]$ in the image plane and the next temporal window size dt_{i+1}.*

Given the solution to Problem 1a and a set of readings from the IMU, our state estimation algorithm in Sec. 3.6 then employs an Extended Kalman Filter with a structureless vision model, as first introduced in [99]. Structured EKF schemes impose vision constraints on all the features between each two subsequent camera poses, and as a result optimize over both the camera poses and the feature positions. A structureless model, on the other hand, allows us to impose constraints between all camera poses that observe each feature, resulting in a state vector containing only the IMU and camera poses. This drastically reduces the size of the state vector, and allows for marginalization over long feature tracks. We pose the state estimation problem as the sub problem below:

Problem 1b (Visual Inertial Odometry)**.** *Given inertial measurements $\mathcal{I}$ at times $\{\tau_k\}$, a camera state s_i at time T_i, a temporal window size dt_i, and feature tracks $\mathcal{F}(t)$ for $t \in [T_i, T_{i+1}]$, estimate the camera state s_{i+1}.*

Algorithm 2 EVIO

```
Input: sensor state $s_i$, events $\mathcal{E}$, IMU readings $\mathcal{I}$, window size $dt_i$
Track features $\{f\}$ in the event stream, $\mathcal{E}$, given $s_i$ (Alg. 3)
Select new features in the image plane (Sec. 3.5.3)
Calculate the size of the next temporal window $dt_{i+1}$ (Sec. 3.5.3)
Update the state estimate $s_{i+1}$ given $\{f\}$ and $\mathcal{I}$ (Alg. 4)
```

3.5 Event-based Feature Tracking

Given a camera state s_i and a temporal window $[T_i, T_{i+1}]$, the goal of Problem 1a is to track a collection of features $\mathcal{F}(t)$ in the image domain for $t \in [T_i, T_{i+1}]$, whose positions are initialized using traditional image techniques (Section 3.5.3). Our approach performs two expectation-maximization (EM) optimizations (Sec. 3.5.1 and Sec. 3.5.2) to align a 3-D spatiotemporal window of events with a prior 2-D template in the image plane.

The feature tracking proposed in this Section is developed from the one proposed in Chapter 2. However, we propose in this work two improvements to the tracker which utilize information from prior iterations and gathered from the IMU, in order to reduce the runtime complexity of the algorithm and improve the quality of the tracking.

The optical flow estimation proposed in this section mirrors the previous method in Section 2.4.1, by solving the data association and optical flow estimation problem jointly with expectation maximization. However, the previous method assumed no knowledge about the image template from which the events are generated. This resulted in an expectation step where data associations had to be computed between every pair of events, which was relatively expensive to compute. In this work, we use the deblurred events from the previous iteration to act as the template, and match the current events to the deblurred events from the previous iteration. By performing decimation to the deblurred events, we are able to reduce the dimensionality of the template significantly, which in turn significantly reduces the number of comparisons needed in the expectation step.

We are also able to improve the affine alignment step, by inferring additional information about this alignment from the rotations estimated from the IMU. We leverage this information as a prior, by derotating the current set of deblurred events by the estimated rotation. This reduces the alignment problem from estimating an affine transform to a scale and translation. This not only reduces the number of parameters that must be estimated, but also reduces the number of local minima during the optimization.

3.5.1 Optical Flow Estimation

The motion of a feature $f(t)$ in the image plane can be described using its *optical flow* $\dot{f}(t)$ as follows:

$$f(t) = f(T_i) + \int_{T_i}^{t} \dot{f}(\zeta) d\zeta = f(T_i) + (t - T_i)u(t) \tag{3.3}$$

where $u(t) := \frac{1}{t-T_i} \int_{T_i}^{t} \dot{f}(\zeta) d\zeta$ is the average flow of $f(s)$ for $s \in [T_i, t]$. If $[T_i, T_{i+1}]$ is sufficiently small, we can *assume* that the average flow u is constant and equal to the average flows of all landmarks L_j whose projections are close to f in the image plane. To formalize this observation, we define a spatiotemporal window around $f(T_i)$ as the collection of events, that propagate backwards to and landmarks whose projections at time T_i are close to the vicinity of $f(T_i)$:

$$W_i := \{(x,t) \in \mathcal{E}, L \in \mathcal{L} \mid \|(x - \bar{t}u) - f(T_i)\| \leq \xi, \tag{3.4}$$

$$\|l - f(T_i)\| \leq \xi, t \in [T_i, T_{i+1}]\} \tag{3.5}$$

where ξ is the window size in pixels, $\bar{t} := t - T_i$, $l := h(L, s)$, as defined in (3.2).

Provided that $[T_i, T_{i+1}]$ is small, the following equality should hold for any event e_k and landmark L_j in W_i:

$$\|(x_k - \bar{t}_k u) - l_j\|^2 \mathbb{1}_{\{\alpha(k)=j\}} = 0 \tag{3.6}$$

where the indicator requires that event k associates with landmark j. However, since the data association α between events and landmarks in unknown, we can hope to satisfy the above requirement only in expectation:

$$\mathbb{E}_{\alpha(k)}\|(x_k - \bar{t}_k u) - l_j\|^2 \mathbb{1}_{\{\alpha(k)=j\}} \tag{3.7}$$

$$= r_{kj}\|(x_k - \bar{t}_k u) - l_j)\|^2 = 0 \tag{3.8}$$

where $r_{kj} := \mathbb{P}(\alpha(k) = j)$. Given the flow u of feature $f(T_i)$, due to the measurement model in (3.2), we model the probability that event e_k was generated from landmark L_j as proportional to the probability density function $\phi((x_k - \bar{t}_k u); l_j, \Sigma)$ of a Gaussian distribution with mean l_j and covariance Σ.

Let $[n_i] := \{1, \ldots, n_i\}$ be an enumeration of the events in W_i. The optical flow constraints in (3.8) and the data association probabilities r_{kj} allow us to estimate the optical flow u of feature $f(T_i)$ as follows:

$$\min_u \sum_{k=1}^{n_i} \sum_{j=1}^{m} r_{kj}\|(x_k - \bar{t}_k u) - l_j)\|^2 \tag{3.9}$$

Unfortunately, the landmark projections $\{l_j\}_{j=1}^m$ needed to compute the data association probabilities are unknown. Instead, we approximate $\{l_j\}_{j=1}^m$ in the above optimization with the set

$$\tilde{l}_j^{i-1} := \{(x + (T_i - t)u_{i-1}) | (x, t) \in W_{i-1}\} \tag{3.10}$$

of *forward-propagated events* from the previous iteration to the current time. This set provides a close approximation to $\{l_j\}_{j=1}^m$ because, as u_{i-1} approaches the true optical flow, (3.6) requires that every point in $\{\tilde{l}_j^{i-1}\}_{j=1}^{n_{i-1}}$ approaches some point in l_j at time T_i, due to continuity in the projected feature trajectories. This leads to the following result for estimating the optical flow of feature $f(T_i)$.

Proposition 1. *Given a sufficiently small temporal window $[T_i, T_{i+1}]$ so that the average optical flow u of a feature $f(t)$, $t \in [T_i, T_{i+1}]$, is approximately constant and a spatiotemporal window W_i of n_i events associated with f, the optical flow u can be estimated by iterating the following EM steps:*

$$\text{E) } r_{kj} = \frac{\phi((x_k - t_k u); \tilde{l}_j^{i-1}, \Sigma)}{\sum_{j'} \phi((x_k - t_k u); \tilde{l}_{j'}^{i-1}, \Sigma)}, \quad \begin{matrix} k \in [n_i] \\ j \in [n_{i-1}] \end{matrix} \tag{3.11}$$

$$\text{M) } u = \frac{\sum_{k=1}^{n_i} \sum_{j=1}^{n_{i-1}} r_{kj}(x_k - \tilde{l}_j^{i-1})\bar{t}_k}{\sum_{k=1}^{n_i} \sum_{j=1}^{n_{i-1}} r_{kj}\bar{t}_k^2} \tag{3.12}$$

Proof. Given an optical flow estimate, the E step computes the likelihood r_{kj} of the data association between events e_k and approximate landmark projections $\tilde{l}_j^{i-1}$ by propagating the events backwards in time and applying the measurement model in (3.2). Given r_{kj}, the M step is a weighted linear least squares problem in u, which corresponds to the overdetermined system $ud^T = Y$, where:

$$d := \left[\sqrt{r_{12}}\bar{t}_1, \dots, \sqrt{r_{kj}}\bar{t}_k, \dots\right]^T \tag{3.13}$$

$$Y := \left[\sqrt{r_{12}}(x_1 - \tilde{l}_2^{i-1}), \dots, \sqrt{r_{kj}}(x_k - \tilde{l}_j^{i-1}), \dots\right] \tag{3.14}$$

To get the normal equations, we multiply both sides on the right by d and obtain $u = (Yd)/(d^T d)$. $\square$

During the initialization of each feature, no prior flow is known, and so we further substitute the prior events in $\{\tilde{l}_j\}$ with the current events and flow, $\{(x - \bar{t}u), (x, t) \in W_i\}$ [166]. Once again, this approximation approaches the true projected landmark positions as u approaches the true flow. The M step, in this case, becomes:

$$u = \frac{\sum_{k=1}^{n_i} \sum_{j=1}^{n_1} r_{kj}(x_k - x_j)(\bar{t}_k - \bar{t}_j)}{\sum_{k=1}^{n_i} \sum_{j=1}^{n_i} r_{kj}(\bar{t}_k - \bar{t}_j)^2} \tag{3.15}$$

where r_{kj} is computed as in (3.11). This method is significantly slower, as the distance computation in the E step uses two different sets of points every iteration. This is detrimental for most neighbor search data structures such as k-d trees, as the data structure must be reconstructed at every iteration. However, as this step is only performed once per feature, it is effectively marginalized out for long feature tracks. We refer to [166] for the full proof of this initialization step.

3.5.2 Template Alignment and RANSAC

While Prop. 1 is sufficient to solve Problem 1a, the feature position estimates may drift as a feature is being tracked over time, due to noise in each flow estimate. To correct this drift, we estimate it as the affine warping that warps $\{\tilde{l}_k^i\}_{k=1}^{n_i}$ (3.10) to the template $\{\tilde{l}_j^{i*}\}_{j=1}^{n_{i*}}$ in the first camera pose that observed the feature. We assume that the corresponding landmarks $\{l\}$ are planar in 3-D, so that we can alternatively represent the affine wrapping as a 3-D rotation and scaling. The 3-D rotation ${}^{i*}R_i$ from the current camera pose at T_i to the first camera pose at T_{i*} can be obtained from the filter used to solve Problem 1b (see Sec. 3.6). Hence, in this section we focus on estimating only a scaling σ and translation b between $\{\tilde{l}_k^i\}$ and $\{\tilde{l}_j^{i*}\}$. First, we rotate each point $\tilde{l}_k^i$ to the first camera frame and center at the rotated feature position as follows:

$$y_k^i = \pi\left({}^{i*}R_i \begin{pmatrix} l_k^i \\ 1 \end{pmatrix}\right) - \pi\left({}^{i*}R_i \begin{pmatrix} f(T_i) + u_i dt_i \\ 1 \end{pmatrix}\right) \tag{3.16}$$

where π is the projection function defined in (3.2). Note that $\tilde{l}_k^i$ propagates events to time T_{i+1}, and so we substitute $f(T_i) + u_i dt_i$ as an estimate for $f(T_{i+1})$. Then, using the same idea as in Sec. 3.5.1, we seek the scaling σ and translation b that minimize the mismatch between $\{y_k^i\}_{k=1}^{n_i}$ and $\{\tilde{l}_j^{i*}\}_{j=1}^{n_{i*}}$:

$$\min_{\sigma,b} \sum_{k=1}^{n_i} \sum_{j=1}^{n_{i*}} r_{kj} \|\sigma y_k^i - b - \tilde{l}_j^{i*}|^2 \tag{3.17}$$

This optimization problem has a similar form to problem (3.9) and, as before, can be solved via the following EM steps:

$$\begin{aligned}
&\text{E)} \quad r_{kj} = \frac{\phi(y_k; \tilde{l}_j^{i*}, \Sigma)}{\sum_{j'} \phi(y_k; \tilde{l}_{j'}^{i*}, \Sigma)}, \; k \in [n_i], j \in [n_{i*}] \\
&\text{M)} \quad \begin{cases}
\bar{y} := \dfrac{1}{n_i} \displaystyle\sum_{k=1}^{n_i} y_k \quad \bar{\tilde{l}} := \dfrac{1}{n_{i*}} \displaystyle\sum_{j=1}^{n_{i*}} \tilde{l}_j \\
\sigma = \sqrt{\dfrac{\sum_{k=1}^{n_i} \sum_{j=1}^{n_{i*}} r_{kj} (y_k - \bar{y})^T (\tilde{l}_j - \bar{\tilde{l}})}{\sum_{k=1}^{n_i} \sum_{j=1}^{n_{i*}} r_{kj} (y_k - \bar{y})^T (y_k - \bar{y})}} \\
b = \dfrac{\sigma}{n_i} \displaystyle\sum_{k=1}^{n_i} \sum_{j=1}^{n_{i*}} r_{kj} y_k - \dfrac{1}{n_{i*}} \displaystyle\sum_{k=1}^{n_i} \sum_{j=1}^{n_{i*}} r_{kj} \tilde{l}_j \\
\quad = \dfrac{\sigma}{n_i} \displaystyle\sum_{k=1}^{n_i} y_k - \dfrac{1}{n_{i*}} \displaystyle\sum_{k=1}^{n_i} \sum_{j=1}^{n_{i*}} r_{kj} \tilde{l}_j \\
\text{as} \displaystyle\sum_{j=1}^{n_{i*}} r_{kj} = 1
\end{cases}
\end{aligned} \tag{3.18}$$

where the M step is solved as in scaled ICP [177].

3.5.3 Feature Detection and Temporal Window Selection

Given the feature tracks after the alignment step, we estimate the size of the next temporal window using the event lifetimes as in Section 2.4.4. We also perform feature detection in a similar method as Section 2.4.5 over event images. However, rather than finding Harris corners and applying non-maximum suppression, we split the image into a predetermined grid, and detect FAST features within each grid cell. This allows for our features to be evenly distributed throughout the image, which is crucial for accurate odometry estimates [139].

3.5.4 Outlier Rejection

In order to remove outliers from the above optimizations, only pairs of points $((x_k - \bar{t}_k u)$ and $(\sigma y_k - b))$, and approximate projected landmarks, $\tilde{l}_j$, with Mahalanobis distance[4] below a set threshold are used in the optimizations. This also serves the purpose of heavily reducing the number of computation.

After all of the features have been updated, two-point RANSAC [142] is performed given the feature correspondences and the rotation between the frames from the state to remove features whose tracking have failed. Given two correspondences and the rotation, we estimate the essential matrix, and evaluate the Sampson error[5] on the set of correspondences to determine the inlier set.

The complete feature tracking procedure is illustrated in Figure 5 and summarized in Algorithm 3.

3.6 State Estimation

To estimate the 3D pose of the camera over time, we employ an Extended Kalman Filter with a structureless vision model, as first developed in [99]. For compactness, we do not expand on the fine details of the filter, and instead refer interested readers

[4]The Mahalanobis distance between a point x and a distribution with mean μ and standard distribution Σ is defined as: $d := \sqrt{(x-\mu)^T \Sigma (x-\mu)}$.

[5]Given a correspondence between two points x_1, x_2 and the camera translation t and rotation R between the points, the Essential matrix is defined as: $E = t \times R$, and the Sampson error is defined as: $e = \frac{x_2^T E x_1}{(Ex_1)_1^2 + (Ex_1)_2^2 + (Ex_2)_1^2 + (Ex_2)_2^2}$

Algorithm 3 Event-based Feature Tracking

```
Input
    sensor state $s_i$, current time $T_i$, window size $dt_i$,
    events $\mathcal{E}$ for $t \in [T_i, T_i + dt_i]$,
    features $\{f\}$ and associated templates $\{\tilde{l}^{i-1}\}$, $\{\tilde{l}^{i*}\}$
Tracking
    for each feature $f$ do
        Find events within $W_i$ (3.5)
        cost $\leftarrow \infty$
        while cost $> \epsilon_1$ do
            Update $r_{kj}$ (3.11), $u$ (3.12) and cost (3.9)
        end while
        Back-propagate events to $T_i$ using $u$
        cost $\leftarrow \infty$
        while cost $> \epsilon_2$ do
            Update $r_{kj}$ (3.18), $(\sigma, b)$ (3.18) and cost (3.17)
        end while
        $f \leftarrow f - b + dt_i u$
    end for
    $dt_{i+1} \leftarrow 3/\text{median}(\{\|u\|\})$ (Sec. 3.5.3)
    return $\{f\}$ and $dt_{i+1}$
```

to [98] and [99]. At time T_i, the filter tracks the current sensor state (3.1) as well as all past camera poses that observed a feature that is currently being tracked. The full state, then, is:

$$S_i := S(T_i) = \begin{bmatrix} s_i^T & \bar{q}(T_{i-n})^T & p(T_{i-n})^T & \dots & \bar{q}(T_i)^T & p(T_i)^T \end{bmatrix}^T \tag{3.19}$$

where n is the length of the oldest tracked feature.

Between update steps, the prediction for the sensor state is propagated using the IMU measurements that fall in between the update steps. Note that, due to the high temporal resolution of the event based camera, there may be multiple update steps in between each IMU measurement. In that case, we use the last IMU measurement to perform the propagation.

Given linear acceleration a_k and angular velocity ω_k measurements, the sensor state is propagated using 5th order Runge-Kutta integration:

$$\begin{aligned} \dot{\bar{q}}(\tau_k) &= \frac{1}{2}\Omega(\omega_k - \hat{b}_g(\tau_k))\bar{q}(\tau_k) & \dot{b}_a(\tau_k) &= 0 \\ \dot{p}(\tau_k) &= v(\tau_k) & & \\ \dot{v}(\tau_k) &= R(\bar{q}(\tau_k))^T(a_k - \hat{b}_a(\tau_k)) + g & \dot{b}_g(\tau_k) &= 0 \end{aligned} \tag{3.20}$$

To perform the covariance propagation, we adopt the discrete time model and covariance prediction update presented in [60].

When an update from the tracker arrives, we augment the state with a new camera pose at the current time, and update the covariance using the Jacobian that maps the IMU state to the camera state.

We then process any discarded features that need to be marginalized. For any such feature f_j, the 3D position of the feature $\hat{F}_j$ can be estimated using its past observations and camera poses by Gauss Newton optimization, assuming the camera poses are known [30]. The projection of this estimate into a given camera pose can then be computed using (3.2). The residual, $r^{(j)}$, for each feature at each camera pose is the difference between the observed and estimated feature positions. We then left multiply $r^{(j)}$ by the left null space, A, of the feature Jacobian, H_F, as in [99], to eliminate the feature position up to a first order approximation:

$$
\begin{aligned}
r_0^{(j)} &= A^T r^{(j)} && (3.21)\\
&\approx A^T H_S^{(j)} \tilde{S} + A^T H_F^{(j)} \tilde{F}_j + A^T n^{(j)} := H_0^{(j)} \tilde{S} + n_0^{(j)} && (3.22)\\
r_n &= Q_1^T r_0 && (3.23)
\end{aligned}
$$

$$
H_0 = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} T_H \\ 0 \end{bmatrix} \tag{3.24}
$$

The elimination procedure is performed for all features, and the remaining uncorrelated residuals, $r_0^{(j)}$ are stacked to obtain the final residual r_0. As in [99], we perform one final step to reduce the dimensionality of the above residual. Taking the QR decomposition of the matrix H_0, we can eliminate a large part of the residual (3.23). The EKF update step is then $\Delta S = K r_n$, where K is the Kalman gain.

When a feature track is to be marginalized, we apply a second RANSAC step to find the largest set of inliers that project to the same point in space, based on reprojection error. This removes moving objects and other erroneous measurements from the track.

Algorithm 4 State Estimation

```
Input
    sensor state $s_i$, features $\{f\}$
    IMU values $\mathcal{I}$ for $t \in [T_i, T_i + dt_i]$
Filter
    Propagate the sensor state mean and covariance (3.20)
    Augment a new camera state
    for each filter track to be marginalized do
        Remove inconsistent observations
        Triangulate the feature using GN Optimization
        Compute the uncorrelated residuals $r_0^{(j)}$ (3.22)
    end for
    Stack all of the $r_0^{(j)}$
    Perform QR decomposition to get the final residual (3.23)
    Update the state and state covariance
```

3.6.1 Feature Track Consistency

Our EKF model assumes a static scene, and cannot explicitly handle features moving independently from the camera. While the first RANSAC step may catch such motions, the interframe motion is generally too small to distinguish between a noisy observation and an independently moving feature. To avoid marginalizing such features, we employ a second RANSAC step within each feature track as it is marginalized, to find the largest inlier set of feature observations that triangulates to a single point in 3D space. In particular, we aim to find, for a given triangulation of a feature from two observations, the largest set of observations whose reprojection error from this triangulation is below a given threshold.

3.7 Experiments

We evaluate the accuracy of our filter on the Event-Camera Dataset [100]. The Event-Camera Dataset contains many sequences captured with a DAVIS-240C camera with information from the events, IMU and images. A number of sequences were also captured in an indoor OptiTrack environment, which provides 3D groundtruth pose.

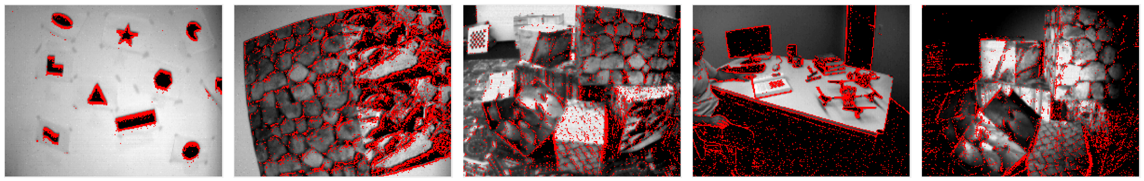


Figure 10: Example images from the Event-Camera Dataset [100] with overlaid events from each sequence. Left to right: shapes, poster, boxes, dynamic, HDR.

Throughout all experiments, dt_0 is initialized as the time to collect 50000 events. The covariance matrices for the two EM steps are set as $2I$, where I is the identity matrix. In each EM step, the template point sets $\{\tilde{l}_j\}$ are subsampled using sphere decimation [48], with radius 1 pixel. As both sets of templates remain constant in both EM steps, we are able to generate a k-d tree structure to perform the Mahalanobis distance checks and E-Steps, generating a significant boost in speed. The Mahalanobis threshold was set at 4 pixels.

At present, our implementation of the feature tracker in C++ is able to run in real time for up to 15 features for moderate optical flows on a 6 core Intel i7 processor. The use of a prior template for flow estimation, and 3D rotation for template alignment has allowed for very significant improvements in runtime, as compared to [166]. In these experiments, 100 features with spatial windows of 31x31 are tracked. Unfortunately, as we must process a continuous set of time windows, there is no equivalent of lower frame rates, so tracking a larger number of features results in slower than realtime

	EVIO		KLTVIO	
Sequence	Mean Position Error (%)	Mean Rotation Error (deg/m)	Mean Position Error (%)	Mean Rotation Error (deg/m)
shapes_translation	2.42	0.52	**1.98**	**0.04**
shapes_6dof	**2.69**	0.40	8.95	**0.06**
poster_translation	**0.94**	0.02	0.97	**0.01**
poster_6dof	3.56	0.56	**2.17**	**0.08**
hdr_poster	**2.63**	0.11	2.67	**0.09**
boxes_translation	2.69	0.09	**2.28**	**0.01**
boxes_6dof	3.61	0.34	**2.91**	**0.03**
hdr_boxes	**1.23**	**0.05**	5.65	0.11
dynamic_translation	**1.90**	**0.02**	2.12	0.03
dynamic_6dof	**4.07**	0.56	4.49	**0.05**

Table 1: Comparison of average position and rotation error statistics between EVIO and KLTVIO across all sequences. Position errors are reported as a percentage of distance traveled. Rotation errors are reported in degrees over distance traveled.

performance as of now. However, we believe, with further optimization, that this algorithm can run in real time.

For our evaluation, we examine only sequences with ground truth and IMU (examples in Fig. 10), omitting the outdoor and simulation sequences. In addition, we omit the rotation only sequences, as translational motion is required to triangulate points between camera poses.

We compare our event based tracking algorithm with a traditional image based algorithm by running the KLT tracker on the images from the DAVIS camera, and passing the tracked features through the same EKF pipeline (we will call this KLTVIO). The MATLAB implementation of KLT is used.

In Fig. 11, we show the temporal size at each iteration. Note that the iterations near the end of the sequence are occurring at roughly 100Hz.

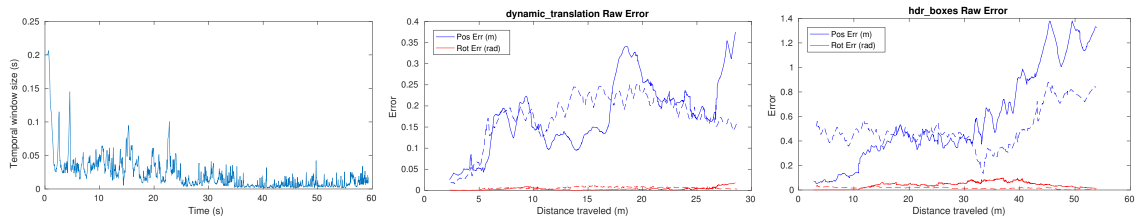


Figure 11: Left to right: Temporal window sizes in the hdr boxes sequence, absolute position and rotation errors for the dynamic translation and hdr boxes sequences. EVIO results are solid, while KLT results are dashed.

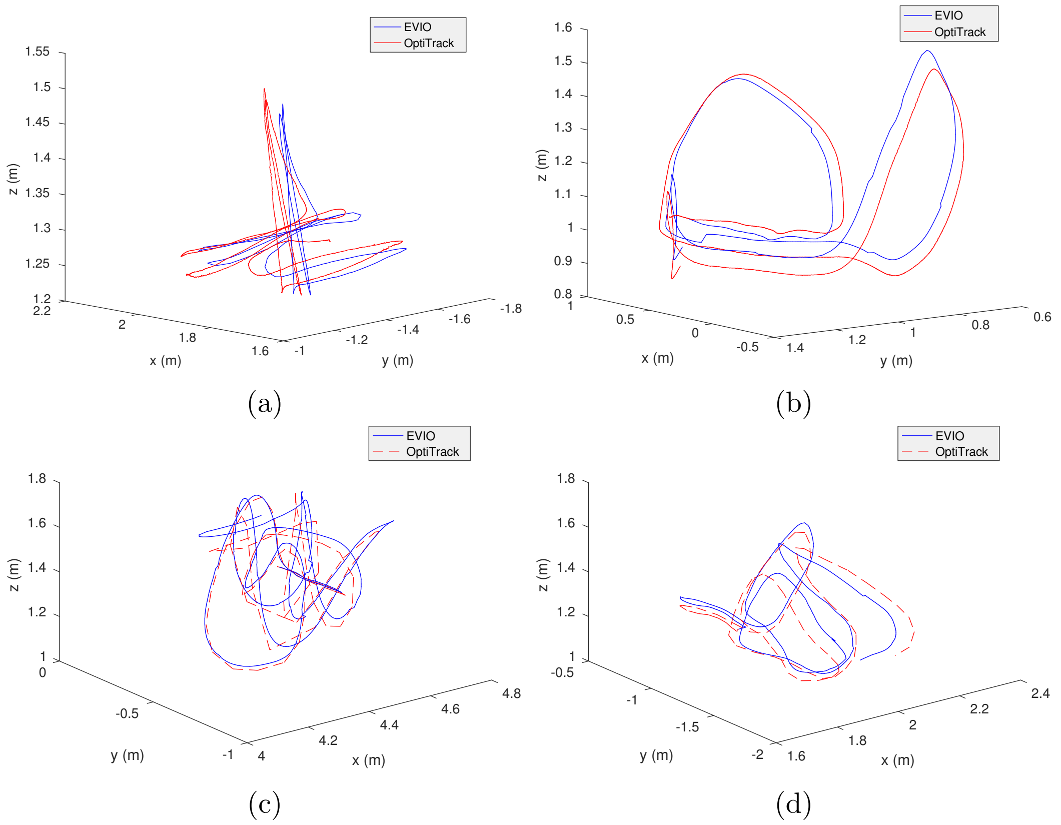


Figure 12: Sample tracked trajectories. (a) shapes translation (b) hdr boxes (c) poster 6dof (d) dynamic 6dof. The first 20 seconds of each sequence are shown, to avoid clutter as the trajectories tends to overlap.

For quantitative evaluation, we compare the position and rotation estimates from both EVIO and KLTVIO against the ground truth provided. Given a camera pose estimate at time T_i, we linearly interpolate the pose of the two nearest OptiTrack measurements to estimate the ground truth pose at this time. Position errors are computed using Euclidean distance, and rotation errors are computed as $err_{rot} = \frac{1}{2}\mathrm{tr}(I_3 - R_{EVIO}^T R_{OptiTrack})$. Sample results for the dynamic_translation and hdr_boxes sequences are shown in Fig. 11.

In Fig. 12, we also show two sample trajectories tracked by our algorithm over a 15 second period, where we can qualitatively see that the estimated trajectory is very similar to the ground truth.

In Table 1, we present the mean position error as a percentage of total distance traveled and rotation error over distance traveled for each sequence, which are common metrics for VIO applications [48].

3.8 Discussion

From the results, we can see that our method performs comparably to the image based method in the normal sequences. In particular, EVIO outperforms KLTVIO in sequences where event-based cameras have an advantage, such as with high speed motions in the dynamic sequences and the high dynamic range scenes. On examination of the trajectories, our method is able to reconstruct the overall shape, albeit with some drift, while the KLT based method is prone to errors where the tracking completely fails. Unfortunately, as the workspace for the sequences is relatively small (3m x 3m), it is difficult to distinguish between drift and failure from error values alone.

In Table 1, we can see that our rotation errors are typically much larger than expected. On inspection of the feature tracks (please see the supplemental video), we can see that, while the majority of the feature tracks are very good, and most failed tracks are rejected by the RANSAC steps, there are still a small, but significant portion of feature tracks that fail, but are not immediately rejected by RANSAC. This equates to the feature tracker estimating the correct direction of the feature's motion (i.e. along the epipolar line), but with an incorrect magnitude. Unfortunately, this is an error that two-point RANSAC cannot resolve. In addition, the error in EM2 can also fail to detect a failed track, for example in situations where the template points are very dense, and so every propagated event is close to a template point, regardless of the scaling and translation. As the EKF is a least squares minimizer, the introduction of any such outlier tracks has a significant effect on the state estimate. However, overall, our event based feature tracking is typically able to track more features over longer periods of time than the image based technique, and we believe that we should be able to remove these outliers and significantly reduce the errors by applying more constraints to the feature motion in future works. These outliers are less common in KLTVIO, as it has been tuned to be over conservative when rejecting feature motions (although this results in shorter feature tracks, overall).

In addition, one of the main challenging aspects of this dataset stems from the fact that, as event-based cameras tend to only trigger events over edge-like features, low texture areas generate very few events, if any. In many of the sequences, the camera passes over textureless areas, such as the back wall of the room, resulting in no events generating in the parts of the image containing the wall. As a result, no features are tracked over these areas. When these areas in the image are large, as in the examples in Figure 13, this introduces biases into the filter pose estimate, and

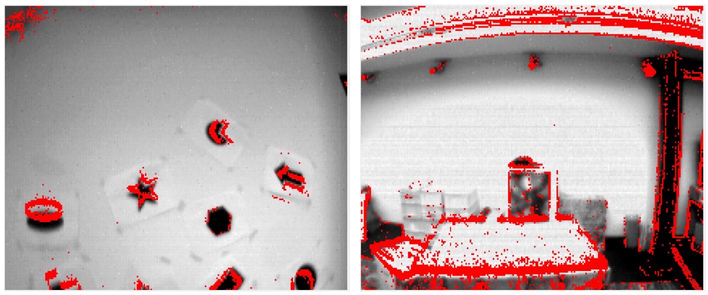


Figure 13: Challenging situations with events within a temporal window (red) overlaid on top of the intensity image. From left to right: boxes_6dof sequence: majority back wall with no events. shapes_6dof: events only generate on edges of a sparse set of shapes, with portions also mostly over a textureless wall

increases tracking error. This typically tends to affect EVIO more than KLTVIO, as areas where no events generate may still have some texture, with the exception being the shapes_6dof sequence, where the KLT tracker fails at the beginning. As a result, this lack of information is a significant contributor to feature track failure.

3.9 Conclusions

In this chapter, we have presented a novel event-based visual inertial odometry algorithm that uses event based feature tracking with probabilistic data associations and an EKF with a structureless vision model. We show that our work is comparable to vision based tracking algorithms, and that it is capable of tracking long camera trajectories with a small amount of drift. Future work involves bringing an implementation of this method to real time, and placing it in a feedback control loop of a robot.

Chapter 4

The Multi Vehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception

4.1 Introduction

After developing the algorithms in Chapters 2 and 3, it became evident that having access to high quality data and ground truth for the target task is crucial to research and development. The Event Camera Dataset [100] provided a useful workbench for pose estimation style tasks, but more data was needed for other tasks, such as optical flow and depth estimation. At the time, a number of papers had been published with event data in a variety of scenes, as described in Section 4.2.1. However, these datasets typically either had a limited range of motions, environments, or ground truth available.

In this chapter, we will outline the Multi Vehicle Stereo Event Camera dataset. This dataset was collected with the goal of being general, with sequences in a variety of different environments, and many ground truth measurements suitable for 3D perception tasks. In providing such a general dataset, we hoped to provide a suitable baseline for developing event camera algorithms that generalize to the complexities of the real world. Another main contribution of this work was as the first dataset with a synchronized stereo event camera system. A calibrated stereo system is useful for depth estimation with metric scale, which can contribute to problems such as pose estimation, mapping, obstacle avoidance and 3D reconstruction. There have been

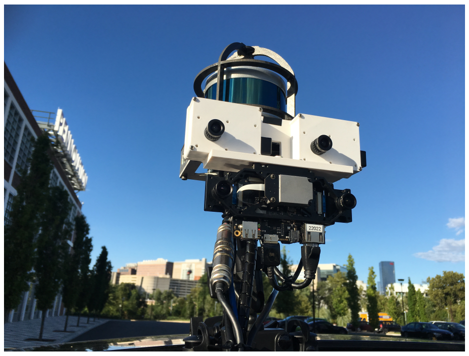


Figure 14: Full sensor rig, with stereo DAVIS cameras, VI Sensor and Velodyne lidar.

a few works in stereo depth estimation with event based cameras, but, due to the lack of accurate ground truth depth, the evaluations have been limited to small, disparate sequences, consisting of a few objects in front of the camera. In comparison, this dataset provides event streams from two synchronized and calibrated Dynamic Vision and Active Pixel Sensors [18] (DAVIS-346b), with long indoor and outdoor sequences in a variety of illuminations and speeds, along with accurate depth images and pose at up to 100Hz, generated from a lidar system rigidly mounted on top of the cameras, as in Fig 14, along with motion capture and GPS.

The full dataset can be found online at `https://daniilidis-group.github.io/mvsec`.

The main contributions from this paper can be summarized as:

- The first dataset with synchronized stereo event cameras, with accurate ground truth depth and pose.
- Event data from a handheld rig, a flying hexacopter, a car, and a motorcycle, in conjunction with calibrated sensor data from a 3D lidar, IMUs and frame based images, from a variety of different speeds, illumination levels and environments.

4.2 Related Work

4.2.1 Related Datasets

At present, there are a number of existing datasets that provide events from monocular event based cameras in conjunction with a variety of other sensing modalities and ground truth measurements that are suitable for testing a number of different 3D perception tasks.

Weikersdorfer et al. [151] combine the earlier eDVS sensor with 128x128 resolution, with a Primesense RGBD sensor, and provide a dataset of indoor sequences with ground truth pose from a motion capture system, and depth from the RGBD sensor.

Rueckauer et al. [130] provide data from a DAVIS 240C camera undergoing pure rotational motion, as well as ground truth optical flow based on the angular velocities reported from the gyroscope, although this is subject to noise in the reported velocities.

Barranco et al. [8] present a dataset with a DAVIS 240B camera mounted on top of a pan tilt unit, attached to a mobile base, along with a Microsoft Kinect sensor. The dataset provides sequences of the base moving with 5dof in a indoor environment, along with ground truth depth, and optical flow and pose from the wheel encoders on the base and the angles from the pan tilt unit. While the depth from the Kinect is accurate, the optical flow and pose are subject to drift from the position estimates of the base's wheel encoders.

Mueggler et al. [100] provide a number of handheld sequences intended for pose estimation in a variety of indoor and outdoor environments, generated from a DAVIS

240C. A number of the indoor scenes have provided pose ground truth, captured from a motion capture system. However, there are no outdoor sequences, or other sequences with a significant displacement, with ground truth information.

Binas et al. [17] provide a large dataset of a DAVIS 346B mounted behind the windshield of a car, with 12 hours of driving, intended for end to end learning of various driving related tasks. The authors provide a number of auxiliary measurements from the vehicle, such as steering angle, accelerator pedal position, vehicle speed etc., as well as longitude and latitude from a GPS unit. However, no 6dof pose is provided, as only 2D translation can be inferred from the GPS output as provided.

These datasets provide valuable data for development and evaluation of event based methods. However, they have, to date, only monocular sequences, with ground truth 6dof pose limited to small indoor environments, with few sequences with ground truth depth. In contrast, this work provides stereo sequences with ground truth pose and depth images in a variety of indoor and outdoor settings.

4.2.2 Event Based 3D Perception

Early works in [75], [76] present stereo depth estimation results with a number of spatial and temporal costs. Later works in [114], [39] and [115] have adapted cooperative methods for stereo depth to event based cameras, due to their applicability to asynchronous, point based measurements. Similarly, [128] and [25] apply a set of temporal, epipolar, ordering and polarity constraints to determine matches, while [22] compare this with matching based on the output of a bank of orientation filters. The authors in [11] show a new method to determine the epipolar line, applied to stereo matching. In [178], the authors propose a novel context descriptor to perform matching, and the authors in [133] use a stereo event camera undergoing pure rotation to perform depth estimation and panoramic stitching.

There are also a number of works on event based visual odometry and SLAM problems. The authors in [167] and [140] proposed novel methods to perform feature tracking in the event space, which they extended in [78] and [169] to perform visual and visual inertial odometry, respectively. In [151], the authors combine an event based camera with a depth sensor, to perform visual odometry and SLAM. The authors in [43] use events to estimate angular velocity of a camera, while [74] and [119] perform visual odometry by building an up to a scale map. In addition, [121] and [102] also fuse events with measurements from an IMU to perform visual inertial odometry.

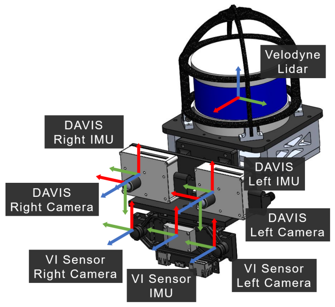
While the more recent works evaluate based on public datasets such as [100], the majority are evaluated on small datasets generated solely for the paper, making comparisons of performance difficult. This is particularly the case for stereo event based cameras. In this work, we try to generate more extensive ground truth, for more meaningful evaluations of new algorithms that can provide a basis for comparisons between methods.

4.3 Dataset

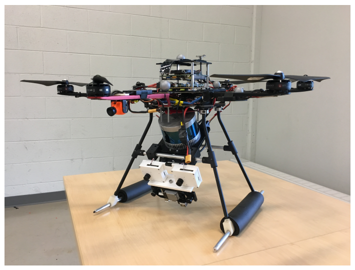
For each sequence in this dataset, we provide the following measurements in ROS bag[1] format:

- Events, APS grayscale images and IMU measurements from the left and right DAVIS cameras.
- Images and IMU measurements from the VI Sensor.

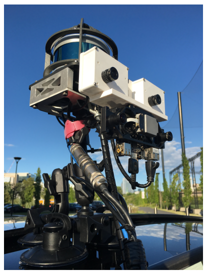
[1] `http://wiki.ros.org/Bags`



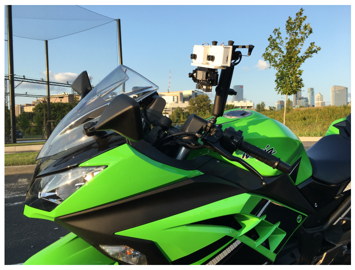
(a)



(b)



(c)



(d)

Figure 15: Examples of sensor configurations. (a): CAD model of the sensor rig. All sensor axes are labeled and colored R:X, G:Y, B:Z, with only combinations of approximately 90 degree rotations between each pair of axes. (b): Sensor package mounted on hexacopter. (c): Sensor package mounted using a glass suction tripod mount on the sunroof of a car. (d): DAVIS cameras and VI Sensor mounted on motorcycle. Note that the VI-Sensor is mounted upside down in all configurations. Best viewed in color.

- Pointclouds from the Velodyne VLP-16 lidar.[2]
- Ground truth reference poses for the left DAVIS camera.
- Ground truth reference depth images for both left and right DAVIS cameras.

4.3.1 Sensors

A list of sensors and their characteristics can be found in Table 2. In addition, Fig 15a shows the CAD drawing of the sensor rig, with all sensor axes labeled, and Fig 15 shows how the sensors are mounted on each vehicle. The extrinsics between all sensors are estimated through calibration, as explained in Sec 4.5.

[2]`http://velodynelidar.com/vlp-16-lite.html`

Sensor	Characteristics
DAVIS 346B	346x260 pixel APS+DVS FOV: 67° vert., 83° horiz. IMU: MPU 6150
VI-Sensor	Skybotix integrated VI-sensor stereocamera: 2 Aptina MT9V034 gray 2x752x480 @ 20fps, global shutter FOV: 57° vert., 2 x 80° horiz. IMU: ADIS16488 @200Hz
Velodyne Puck LITE	VLP-16 PUCK LITE 360° Horizontal FOV, 30° Vertical FOV 16 channel 20Hz 100m Range
GPS	UBLOX NEO-M8N 72-channel u-blox M8 engine Position accuracy 2.0 m CEP

Table 2: Sensors and characteristics.

For event generation, two DAVIS-346B cameras are mounted in a horizontal stereo setup. The cameras are similar to [18], but have a higher, 346x260 pixel, resolution, up to 50fps APS (frame based images) output, and higher dynamic range. The baseline of the stereo rig is 10cm, and the cameras are timestamp synchronized by using the trigger signal generated from the left camera (master) to deliver sync pulses to the right (slave) through an external wire. Both cameras have 4mm lenses with approximately 87 degrees horizontal field of view, with an additional IR cut filter placed on each one to suppress the IR flashes from the motion capture systems. The APS exposures are manually set (no auto exposure) depending on lighting conditions, but are always the same between the cameras. While the timestamps of the grayscale DAVIS images are synced, there is unfortunately no way to synchronize the image acquisition itself. Therefore, there may be up to 10ms of offset between the images.

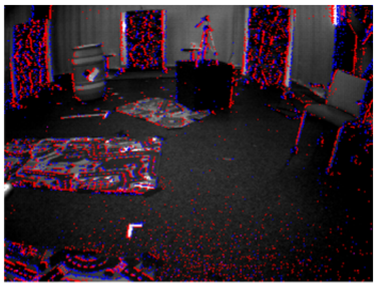
Vehicle	Sequence	T (s)	D (m)	$\|v\|_{\max}$ (m/s)	$\|\omega\|_{\max}$ (°/s)	MER (events/s)	Pose GT	Depth
Hexacopter	Indoor 1*	70	26.7	1.4	28.3	185488	Vicon	Yes
	Indoor 2*	84	36.8	1.5	29.8	273567	Vicon	Yes
	Indoor 3*	94	52.3	1.7	31.0	243953	Vicon	Yes
	Indoor 4*	20	9.8	2.0	64.3	361579	Vicon	Yes
	Outdoor 1	54	33.2	1.5	129.8	261589	Qualisys	Yes
	Outdoor 2	41	29.9	1.5	109.4	256539	Qualisys	Yes
Handheld	In-out	144	80.4	1.6	93.6	468675	LOAM	Yes
	Indoor	249	105.2	2.7	37.4	590620	LOAM	Yes
Car	Day 1†	262	1207	7.6	30.6	386178	Cart., GPS	Yes
	Day 2†	653	3467	12.0	35.5	649081	Cart., GPS	Yes
	Evening 1	262	1217	10.4	20.6	334614	Cart., GPS	Yes
	Evening 2	374	2109	11.2	33.6	404105	Cart., GPS	Yes
	Evening 3	276	1613	10.0	25.1	371498	Cart., GPS	Yes
Motorcycle	Highway 1	1500	18293	38.4	203.4	511024	GPS	No

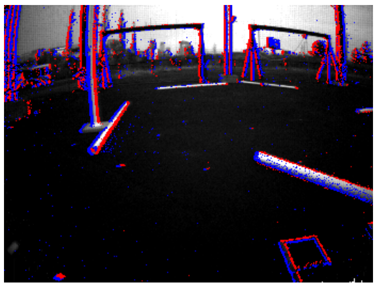
Table 3: Sequences for each vehicle. T: Total time, D: Total distance traveled, $\|v\|_{\max}$: Maximum linear velocity, $\|\omega\|_{\max}$: Maximum angular velocity, MER: Mean event rate.
*No VI-Sensor data is available for these sequences.
†A hardware failure caused the right DAVIS grayscale images to fail for these sequences.

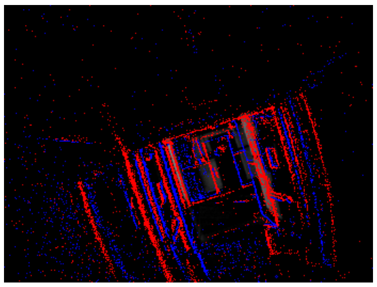
To provide ground truth reference poses and depths (Sec 4.4), we have rigidly mounted a Velodyne Puck LITE above the stereo DAVIS cameras. The Velodyne lidar system provides highly accurate depth of a large number of points around the sensor. The lidar is mounted such that there is full overlap between the smaller vertical field of view of the lidar and that of the stereo DAVIS rig.

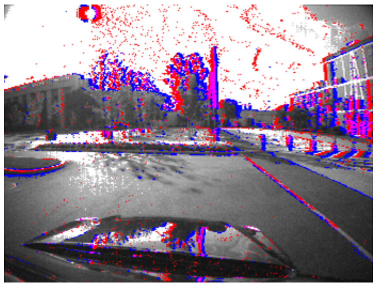
In the outdoor scenes, we have also mounted a GPS device for a second ground truth reference for latitude and longitude. Typically, the GPS is placed away from the sensor rig to avoid interference from the USB 3.0 data cables.

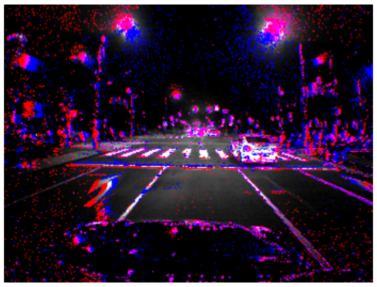
In addition, we have mounted a VI Sensor [108], originally developed by Skybotix for comparison with frame based methods. The sensor has a stereo pair with IMU, all synchronized. Unfortunately, the only mounting option was to mount the cameras upside down, but we provide the transform between them and the DAVIS cameras.


(a) indoor_flying1


(b) outdoor_flying1


(c) indoor1


(d) outdoor_day1.


(e) outdoor_night1

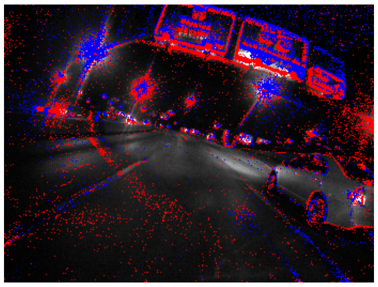

(f) motorcycle

Figure 16: Sample images with overlaid events (blue and red) from indoor and outdoor sequences, during day and evening. Best viewed in color.

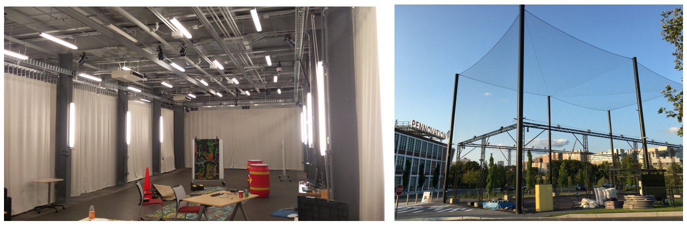


Figure 17: Motion capture arenas. Left: Indoor Vicon arena, right: Outdoor Qualisys arena.

4.3.2 Sequences

The full list of sequences with summary statistics can be found in Table 3, and sample APS images with overlaid events can be found in Fig 16.

Hexacopter with Motion Capture

The sensor setup was mounted below the compute stack of the hexacopter, with a 25 degree downwards pitch, as in Fig 15b. Two motion capture systems are used to generate sequences for this dataset, one indoors and one outdoors (Fig 17). The 26.8m $\times$ 6.7m $\times$ 4.6m indoor area is instrumented with 20 Vicon Vantage VP-16 cameras. The outdoor netted area of 30.5m $\times$ 15.3m $\times$ 15.3m is instrumented with an all-weather motion capture system comprised of 34 high resolution Qualisys Oqus 700 cameras. Both systems provide millimeter accuracy pose at 100Hz by emitting infrared strobes and tracking IR reflecting markers placed on the hexacopter. We provide sequences in each area, with flights of different length and speed.

Handheld

In order to test performance in high dynamic range scenarios, the full sensor rig is carried in a loop through both outdoor and indoor environments, as well as indoor environments with and without external lighting. Ground truth pose and depth is provided by lidar SLAM.

Outdoor Driving

For slow to medium speed sequences, the sensor rig is mounted on the sun roof of a sedan as in Fig 15c, and driven around several West Philadelphia neighborhoods at speeds up to 12 m/s. Sequences are provided in both day and evening situations, including sequences with the sun directly in the cameras' field of view. Ground truth is provided as depth images from a lidar map, as well as pose from loop closed lidar odometry and GPS.

For high speed sequences, the DAVIS stereo rig and VI Sensor are mounted on the handlebar of a motorcycle (Fig 15d), along with the GPS device. The sequences involve driving at up to 38m/s. Longitude and latitude, as well as relative velocity, are provided from the GPS.

4.4 Ground Truth Generation

To provide ground truth poses, motion capture poses are used when available. Otherwise, if lidar is available, Cartographer [61] is used for the driving sequences to fuse the lidar sweeps and IMU data into a loop-closed 2D pose of the lidar, which is transformed into the left DAVIS frame using the calibration in Sec 4.5.4. For outdoor scenes, we also provide raw GPS readings.

For each sequence with lidar measurements, we run the Lidar Odometry and Mapping (LOAM) algorithm [163] to generate dense 3D local maps, which are projected into each DAVIS camera to generate dense depth images at 20Hz, and to provide 3D pose for the handheld sequences.

Two separate lidar odometry algorithms are used as we noted that LOAM produces better, more well aligned, local maps, while Cartographer's loop closure results in more accurate global poses with less drift for longer trajectories. While Cartographer only estimates a 2D pose, we believe that this is a valid assumption as the roads driven have, for the most part, a single consistent grade.

4.4.1 Ground Truth Pose

For the sequences in the indoor and outdoor motion capture arenas, the pose of the body frame of the sensor rig ${}^{\text{world}}\mathbf{H}_{\text{body}(t)}$ at each time t is measured at 100Hz with millimeter level accuracy.

For outdoor sequences we rely on Cartographer to perform loop closure and fuse lidar sweeps and IMU data into a single loop-closed 2D pose of the body (lidar in this case) with minimal drift.

In order to provide a quantitative measure of the quality of the final pose, we align the positions with the GPS measurements, and provide both overlaid on top of satellite imagery for each outdoor sequence in the dataset, as well as the difference in position between the provided ground truth and GPS. Fig 20 provides a sample overlay for Car Day 2, where the average error between Cartographer and the GPS is consistently around 5m without drift. This error is consistent amongst all of the outdoor driving sequences, where the overall average error is 4.7m, and is on a similar magnitude to the error expected from GPS. Note that the spike in error around 440 seconds is due to significant GPS error, and corresponds to the section in bold on the top right of the overlay.

In both cases, the extrinsic transform, represented as a 4×4 homogeneous transform matrix ${}^{\text{body}}\mathbf{H}_{\text{DAVIS}}$, for each sequence that takes a point from the left DAVIS frame to the body frame is then used to estimate the pose of the left DAVIS at time t with respect to the first left DAVIS pose at time t_0:

$$\begin{aligned} {}^{\text{DAVIS}(t_0)}\mathbf{H}_{\text{DAVIS}(t)} = \\ {}^{\text{body}}\mathbf{H}_{\text{DAVIS}}^{-1}\,{}^{\text{world}}\mathbf{H}_{\text{body}(t_0)}^{-1}\,{}^{\text{world}}\mathbf{H}_{\text{body}(t)}\,{}^{\text{body}}\mathbf{H}_{\text{DAVIS}}. \end{aligned} \tag{4.1}$$

4.4.2 Depth Map Generation

In each sequence where lidar is available, depth images for each DAVIS camera are generated for every lidar measurement. We first generate a local map by transforming each lidar pointcloud in a local window around the current measurement into the frame of the current measurement using the poses from LOAM. At each measurement, the window size is determined such that the distances between the current, and the first and last LOAM poses in the window are at least d meters, and that there are at least s seconds between the current, and first and last LOAM poses, where d and s are parameters tuned for each sequence. Examples of these maps can be found in Fig 18.

We then project each point, $\mathbf{p}$, in the resulting pointcloud into the image in each DAVIS camera, using the standard pinhole projection equation:

$$\begin{pmatrix} u & v & 1 \end{pmatrix}^T = \mathbf{K}\Pi\left({}^{\text{body}}\mathbf{H}_{\text{DAVIS}}\begin{bmatrix}\mathbf{p}\\1\end{bmatrix}\right) \tag{4.2}$$

where Π is the projection function:

$$\Pi\left(\begin{pmatrix} X & Y & Z & 1 \end{pmatrix}^T\right) = \begin{pmatrix} \frac{X}{Z} & \frac{Y}{Z} & 1 \end{pmatrix}^T \tag{4.3}$$

and $\mathbf{K}$ is the camera intrinsics matrix for the rectified image (i.e. the top left 3×3 of the projection matrix).

Any points falling outside the image bounds are discarded, and the closest point at each pixel location in the image is used to generate the final depth map, examples of which can be found in Fig 19.

In addition, we also provide raw depth images without any undistortion by unrectifying and distorting the rectified depth images using the camera intrinsics and OpenCV.

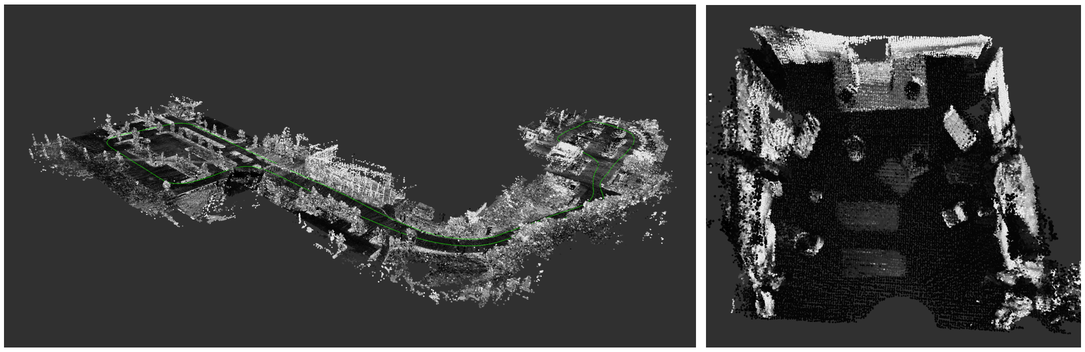


Figure 18: Sample maps generated for ground truth. **Left**: Full map from Car Day 1 sequence, trajectory in green. **Right**: Local map from the Hexacopter Indoor 3 sequence.

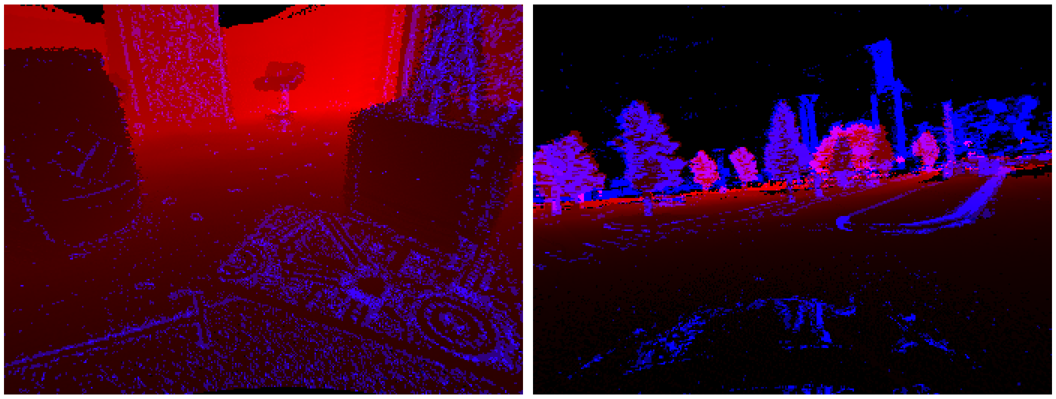
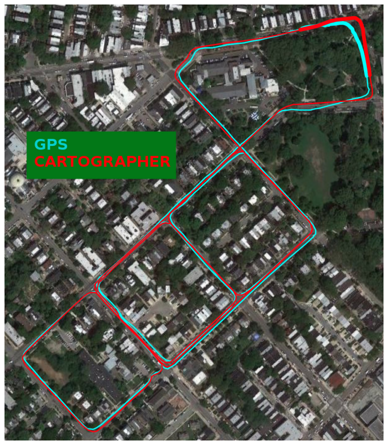


Figure 19: Depth images (red) with events overlaid (blue) from the Hexacopter Indoor 2 and Car Day 1 sequences. Note that parts of the image (black areas, particularly the top) have no depth due to the limited vertical field of view and range of the lidar. These parts are labeled as NaNs in the data. Best viewed in color.

4.5 Calibration

In this section, we describe the various steps performed to calibrate the intrinsic parameters of each DAVIS and VI-Sensor camera, as well as the extrinsic transformations between each of the cameras, IMUs and the lidar. All of the calibration results are provided in yaml form.

The camera intrinsics, stereo extrinsics, and camera-IMU extrinsics are calibrated us-



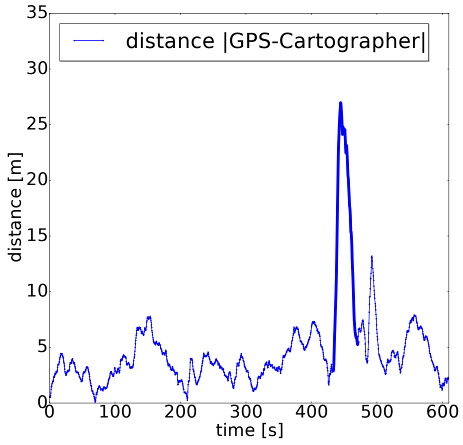


Figure 20: Comparison between GPS and Cartographer trajectories for outdoor_day_2 overlaid on top of satellite imagery. Note that the spike in error between Cartographer and GPS corresponds to the bolded section in the top right of the overlay on the left, and is largely due to GPS error. Best viewed in color.

ing the Kalibr toolbox[3][42], [40], [92], the extrinsics between the left DAVIS camera and Velodyne lidar are calibrated using the Camera and Range Calibration Toolbox[4][49], and fine tuned manually, and the hand eye calibration between the mocap model pose in the motion capture world frame and the left DAVIS camera pose is performed using CamOdoCal[5] [59]. To compensate for changes in the mounted rig, each calibration is repeated each day data was collected, and every time the sensing payload was modified. In addition to the calibration parameters, the raw calibration data for each day is also available on demand for users to perform their own calibration, if desired.

4.5.1 Camera Intrinsic, Extrinsic and Temporal Calibration

The camera intrinsics and extrinsics are estimated using a grid of AprilTags [109] that is moved in front of the sensor rig and calibrated using Kalibr. Each calibration provides the focal length and principal point of each camera as well as the distortion

[3]`https://github.com/ethz-asl/kalibr`
[4]`http://www.cvlibs.net/software/calibration/`
[5]`https://github.com/hengli/camodocal`

parameters and the extrinsics between the cameras.

In addition, we calibrate the temporal offset between the DAVIS stereo pair and the VI Sensor by finding the temporal offset that maximizes the cross correlation between the magnitude of the gyroscope angular velocities from the IMUs of the left DAVIS and the VI Sensor. The timestamps for the VI Sensor messages in the dataset are then modified to compensate for this offset.

4.5.2 Camera to IMU Extrinsic Calibration

To calibrate the transformation between the camera and IMU, a sequence is recorded with the sensor rig moving in front of the AprilTag grid. The two calibration procedures are separated to optimize the quality of each individual calibration. The calibration sequences are once again run through Kalibr using the camera-IMU calibration to estimate the transformations between each camera and each IMU, given the prior intrinsic and camera-camera extrinsic calibrations.

4.5.3 Motion Capture to Camera Extrinsic Calibration

Each motion capture system provides the pose of the mocap model in the motion capture frame at 100Hz. However, the mocap model frame is not aligned with any camera frame, and so a further calibration is needed to obtain the pose of the cameras from the motion capture system.

The sensor rig was statically held in front of an Aprilgrid at a variety of different poses. At each pose at time t_i, the pose of the left DAVIS camera frame in the grid frame ${}^{\text{aprilgrid}}\mathbf{H}_{\text{DAVIS}(t_i)}$, as well as the pose of the mocap model (denoted body) in the mocap frame ${}^{\text{mocap}}\mathbf{H}_{\text{body}(t_i)}$, were measured. These poses were then used to solve the handeye calibration problem for the transform that transforms a point in the left

DAVIS frame into the model frame ${}^{\text{body}}\mathbf{H}_{\text{DAVIS}}$:

$$\begin{aligned} {}^{\text{body}(t_0)}\mathbf{H}_{\text{body}(t_i)} {}^{\text{body}}\mathbf{H}_{\text{DAVIS}} =& {}^{\text{body}}\mathbf{H}_{\text{DAVIS}} {}^{\text{DAVIS}(t_0)}\mathbf{H}_{\text{DAVIS}(t_i)} \\ i =& 1, \dots, n \end{aligned} \tag{4.4}$$

where:

$$ {}^{\text{body}(t_0)}\mathbf{H}_{\text{body}(t_i)} = {}^{\text{mocap}}\mathbf{H}^{-1}_{\text{body}(t_0)} {}^{\text{mocap}}\mathbf{H}_{\text{body}(t_i)} \tag{4.5}$$

$$ {}^{\text{DAVIS}(t_0)}\mathbf{H}_{\text{DAVIS}(t_i)} = {}^{\text{aprilgrid}}\mathbf{H}^{-1}_{\text{DAVIS}(t_0)} {}^{\text{aprilgrid}}\mathbf{H}_{\text{DAVIS}(t_i)}. \tag{4.6}$$

The optimization is performed using CamOdoCal, using the linear method in [33], and refining using a nonlinear optimization as described in [59].

4.5.4 Lidar to Camera Extrinsic Calibration

The transformation that takes a point from the lidar frame to the left DAVIS frame was initially calibrated using the Camera and Range Calibration Toolbox [49]. Four large checkerboard patterns are placed to fill the field of view of the DAVIS cameras, and a single pair of images from each camera is recorded, along with a full lidar scan. The calibrator then estimates the translation and rotation that aligns the camera and lidar observations of the checkerboards.

However, we found that the reported transform had up to five pixels of error when viewing the projected depth images (Fig 19). In addition, as the lidar and cameras are not hardware time synchronized, there was occasionally a noticeable and constant time delay between the two sensors. To improve the calibration, we fixed the translation based on the values from the CAD models, and manually fine tuned the rotation and time offset in order to maximize the overlap between the depth and event images. For visual confirmation, we provide the depth images with events overlaid for each camera. The timestamps of the lidar messages provided in the dataset are compensated for the time offset.

4.6 Known Issues

4.6.1 Moving Objects

The mapping used to generate the depth maps assumes that scenes are static, and typically does not filter out points on moving objects. As a result, the reported depth maps may have errors of up to two meters when tracking points on other cars, etc. However, these objects are typically quite rare compared to the total amount of data available. If desired, future work could involve classifying vehicles in the images and omitting these points from the depth maps.

4.6.2 Clock Synchronization

The motion capture and GPS are only synchronized to the rest of the system using the host computer's time. This may incur an offset between the reported timestamps and the actual measurement time. We record all measurements on one computer to minimize this effect. In addition, there may be some delay between a lidar point's measurement and the timestamp of the message due to the spin rate of the lidar.

4.6.3 DVS Biasing

Default biases for each camera were used when generating each sequence. However, it has been noted that, for the indoor flying sequences, the ratio of positive to negative events is higher than usual ($\sim$2.5-5x). At this point, we are unaware of what may have caused this imbalance, or whether tuning the biases would have balanced it. We note that the imbalance is particularly skewed over the speckled floor. We advise researchers using the polarities of the events to be aware of this imbalance when working with these sequences.

4.7 Conclusions

In this chapter, we have presented a novel dataset for stereo event cameras, on a number of different vehicles and in a number of different environments, with ground truth 6dof pose and depth images. This work is ongoing, and we hope to release more sequences as they are processed. Since publication, this work has helped with the development of a number of projects, and allowed for a principled comparison of novel methods.

Chapter 5

Realtime Time Synchronized Event-based Stereo

5.1 Introduction

Given a stereo pair of event cameras, which are rigidly mounted to each other along the horizontal axis, we can rectify the left and right images. That is, we can rotate them such that the epipolar line in each image is exactly along the x axis. In these rectified images, the problem is to find correspondences in the right image plane in the same row for each point in the left image plane. The distance between these correspondences is known as the disparity. Given these disparity values, it is possible to use the known geometry of the camera configuration to accurately estimate the depth of each point in the scene. Traditional stereo methods capitalize on the photo-consistency assumption, finding correspondences where the image intensities between the left and right correspondences match.

However, as noted before, such an assumption cannot be used for events. A similar problem facing general event-based methods is that of time synchronization. That is, events generated at different times may correspond to the same point in the image space, but will appear at different pixel positions due to the motion of the point. This problem manifests itself in two ways. Between cameras, this problem is analogous to having unsynchronized cameras for frame based stereo methods, where the epipolar constraint breaks down due to the motion between the images, and occurs when events are not generated at the same time between the two cameras. Within a single camera, this causes effects similar to the motion blur seen in frame based images. For the stereo matching problem, which is often solved using appearance based similarity metrics, this blurring is highly detrimental, as it often alters the appearance of each image differently. A number of event based stereo methods have approached these problem with asynchronous methods (e.g. [114, 128, 154]), which process each event independently. However, these methods must either forgo the information provided by the spatial neighborhood around each event, or use fine tuned temporal windows to once again ensure time synchronization, as there are no guarantees that neighboring events were generated at a similar time.

In this chapter, we show that this problem can be resolved for stereo disparity matching if the velocity of the camera is known. In particular, we propose a novel event disparity volume for events from a stereo event camera pair, that uses the motion of the camera to temporally synchronize the events with temporal interpolation at each disparity. Our method utilizes the motion blur idea presented in Section 1.6. To estimate optical flow, we use the motion field equation, given camera velocity and a set of disparities, and similarly interpolate the position of the events at each disparity to a single point in time, which we represent as a novel temporally synchronized event disparity volume. We show that, in addition to removing motion blur at the correct disparities (where the motion field equation is valid), this volume allows us to disambiguate otherwise challenging regions in the image by inducing additional motion blur.

We then define a novel matching cost over this event disparity volume, which rewards similarity between patches, while penalizing blurriness inside the patch. We show that this cost function is able to robustly distinguish the true disparity, while being extremely cheap to compute, using only bitwise comparisons over a sliding window.

Our method, implemented in Tensorflow, runs in realtime at 40ms on a laptop grade GPU, with significant further optimizations available. We evaluate our results on the Multi Vehicle Stereo Event Camera dataset (Chapter 4), and show significant improvements in disparity error over state of the art event based stereo methods, which rely on additional, more computationally expensive, smoothness regularizations.

The main contributions of this chapter are summarized as:

- A novel method for using camera velocity to generate a time synchronized event disparity volume where regions at the correct disparity are in focus, while regions at the incorrect disparity are blurred.
- A novel block matching based cost function over an event disparity volume that jointly rewards similarity between left and right patches while penalizing blurriness in both patches.
- Evaluations on the Multi Vehicle Stereo Event Camera dataset, with comparisons against other state of the art methods, and evaluations of each component of the method.

5.2 Related Work

Early works in stereo depth estimation for event cameras, such as the works by Kogler et al. [76] and Rogister et al. [128], attempted to perform matching between individual events in a fully asynchronous fashion, using a combination of temporal and spatial constraints, which Kogler et al. showed to perform better than basic block matching between pairs of event images. However, these methods suffer from ambiguities in matching when single events are considered.

To address these ambiguities, Camuas-Mesa et al. [22] use local spatial information in the form of local Gabor filters as features, while in [23], they track clusters of events to aid in tracking with occlusion. Zou et al. [178] use a novel local event context descriptor based on the distances between events in a window, which they extend in [179] to produce a dense disparity estimate. Similarly, Schraml et al. [133] use a cost based on the distance between events to generate panoramic depth maps.

In addition, several works have applied smoothing based regularizations to constrain ambiguous regions, which have seen great success in frame based stereo. Piatkowska et al. [114, 115], have applied cooperative stereo methods [91] in an asynchronous fashion, while Xie et al. [154, 155] have adapted belief propagation [14] and semiglobal matching [38], respectively, to similar effect. These regularizations have shown significant improvements over the prior state of the art.

These prior works have all shown promising results for event-based stereo matching, but do not explicitly handle the time synchronization problem without abandoning the rich spatial information around each pixel.

Concurrently with this work, Zhou et al. [165] generates a time surface representation of the events, consisting of the exponential of the differences between subsequent timestamps, and performs a nonlinear optimization over multiple stereo observations, assuming known camera pose, to generate a semi-dense 3D reconstruction.

5.3 Method

The underlying problem of stereo disparity matching can be thought of as a data association problem. That is, to find correspondences between the points in the left and right images, at a given point in time. In this work, we assume that the cameras are calibrated and rectified, so that every epipolar line in both images is parallel to the x axis, and the correspondence problem is reduced to a 1D search along the x dimension. While some prior works such as [128] in the event based literature have tried to perform matching on an event by event basis, we use the spatial neighborhood around each pixel for a more detailed and robust matching, by making a locally constant depth assumption.

It is possible to perform matching on event images generated directly from the event positions. However, such an image generated from the raw events is very prone to motion blur, unless the time window is carefully selected, with a method such as the lifetime estimation in [101].

Motion blur is generated when events are captured at different points in time, such that events corresponding to the same point in the image may occur at different pixels due to the motion of that point. However, the works in [167], [43] and [121] show that motion blur can be removed from an event image if the optical flow for each pixel is known. In Sec. 5.3.1, we leverage this technique to both remove motion blur at the correct disparities, while further blurring the events at incorrect disparities. We then describe a novel event disparity volume representation of these time shifted events in Sec. 5.3.2, on which we apply a novel cost function that leverages this focus-defocus effect to allow us to discriminate the true disparity at each pixel, as described in Sec. 5.3.3. Finally, Sec. 5.3.4 discusses methods to then use the cost function to estimate the true disparity at each pixel.

An overview of the method can be found in Fig. 21

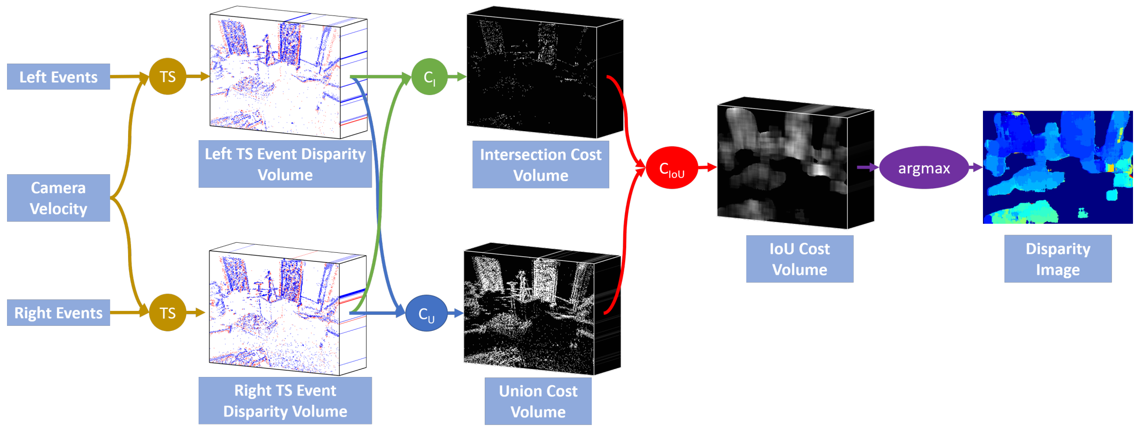


Figure 21: Overview of our method. Given an input of left and right events and camera velocity, left and right time synchronized event disparity volumes are generated (Sec. 5.3.1 and 5.3.2). The intersection and union costs are calculated by combining the two disparity volumes, and the final IoU cost volume is computed (Sec. 5.3.3). Finally, the disparity is computed in a winner takes all scheme over the IoU cost volume (Sec. 5.3.4). Best viewed in color.

5.3.1 Time Synchronization through Interpolation

For a given disparity, d, we can approximate optical flow using the motion field equation, with an assumption of known camera velocity. The motion field equation describes the relationship between the linear ($\mathbf{v}$) and angular ($\boldsymbol{\omega}$) velocity of a camera, depth Z of a point (x, y), which we treat here as a function of disparity, d, and the motion of the point in the image, which we approximate to be the optical flow ($\dot{x}_i$, $\dot{y}_i$):

$$\begin{pmatrix} \dot{x}_i(d) \\ \dot{y}_i(d) \end{pmatrix} = \frac{1}{Z(d)} \begin{bmatrix} -1 & 0 & x_i \\ 0 & -1 & y_i \end{bmatrix} \mathbf{v} + \begin{bmatrix} x_i y_i & -(1+x_i^2) & y_i \\ 1+y_i^2 & -x_i y_i & -x_i \end{bmatrix} \boldsymbol{\omega} \tag{5.1}$$

$$Z(d) = \frac{fb}{d} \tag{5.2}$$

where f is the focal length of the camera and b is the baseline between the two cameras.

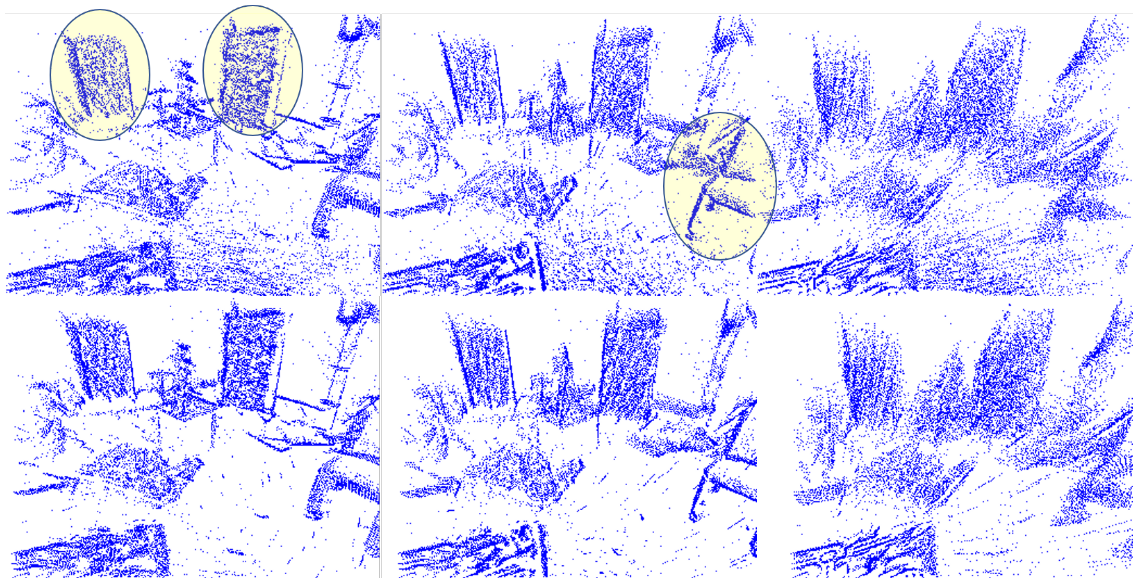


Figure 22: Sample slices of the left (top) and right (bottom) time synchronized event disparity volumes at disparities 6 (left), 16 (middle) and 31 (right). Only positive pixels are shown for clarity. At disparity 6, the boards at the back are in focus, while at disparity 16, the chair in the front is in focus (both circled in yellow). The other features at the wrong depth are blurred. The right slices have been shifted horizontally by the disparity, as in (5.5), so that corresponding points should be at the same x position in both images. Best viewed in color.

Assuming that the optical flow for each pixel is constant within each time window, we can then estimate the position of a point generating an event (x_i, y_i, t_i, p_i) at a constant time t' with linear interpolation:

$$\begin{pmatrix} x'_i(d) \\ y'_i(d) \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} \dot{x}_i(d) \\ \dot{y}_i(d) \end{pmatrix} (t' - t_i) \tag{5.3}$$

Assuming a static scene and accurate velocities and disparities, the set of time synchronized events, $\left\{\begin{pmatrix} x'_i(d) & y'_i(d) & t' & p_i \end{pmatrix}\right\}$, is assumed to have no motion blur for all x_i and y_i with disparity d.

5.3.2 Time Synchronized Event Disparity Volume Generation

However, the true depth for each pixel is unknown for this problem. Instead, we select a range of disparities over which to search, and apply (5.3) to the set of events from the left camera for every disparity within the range.

At each disparity level, d, we generate an image based on the time shifted events, where a pixel with more positive events is set to 1, more negative events is set to -1, and no events is set to 0. Note that the time shifted event positions are rounded to the nearest integer to index into the image.

$$\begin{aligned} I_L(x, y, d) = \text{sign}\left(\sum_i p_i\right) \\ i \in &\{i | (x'_i(d), y'_i(d)) = (x, y)\} \\ p_i \in &\{-1, 1\} \end{aligned} \tag{5.4}$$

This is similar to standard methods that generate images by summing events at each pixel, but the additional sign operator allows the image to be robust to the left or right camera generating more events at each pixel than the other.

The result is a 3D volume for the left camera, where each slice in the disparity dimension represents the images generated according to (5.4), using the disparity corresponding to that slice. That is, when the camera moves with some linear velocity, this flow would have a deblurring effect on points where the pixel position matches the disparity, and potentially apply further blurring on points where the disparity is incorrect. In the case when the camera's motion is pure rotation, the flow will produce deblurred images at each disparity slice.

We apply a similar operation to the events from the right camera, except that the x position of each shifted event is further shifted by the disparity at each level:

$$\begin{aligned} I_R(x, y, d) = \text{sign}\left(\sum_i p_i\right) \qquad (5.5) \\ i \in & \{i | (x'_i(d) + d, y'_i(d)) = (x, y)\} \\ p_i \in & \{-1, 1\} \end{aligned}$$

This generates a set of disparity volumes similar to traditional plane sweep volumes, where the potential matching right pixel corresponding to $I_L(x, y, d)$ is $I_R(x, y, d)$. We show some example slices of this volume in Fig. 23, where the blurring and deblurring effects can be clearly seen.

5.3.3 Matching Cost

Finally, we apply a novel sliding window matching cost that leverages both the deblurring and blurring effects of Sec. 5.3.1. First, it penalizes windows with many events, as this would indicate areas with an incorrect disparity due to the blurring incurred by the temporal interpolation. Given a local spatial window $W(x, y, d)$ around a pixel (x, y) at a given disparity d, we encode this using a union term, defined as:

$$C_U(x, y, d) = \sum_{x^*, y^* \in W(x,y,d)} I_L(x^*, y^*, d) \cup I_R(x^*, y^*, d) \qquad (5.6)$$

$$a \cup b = \begin{cases} 1 & a \neq 0 \text{ or } b \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

We carefully choose the union operator instead of addition, in order to not double penalize pixels with events in both volumes.

Second, the cost rewards windows that are similar. That is, we would like pixels between the two images to have events with the same polarity. We encode this using an intersection term, defined as:

$$C_I(x,y,d) = \sum_{x^*,y^* \in W(x,y,d)} I_L(x^*,y^*,d) \cap I_R(x^*,y^*,d) \tag{5.7}$$

$$a \cap b = \begin{cases} 1 & a = b \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

This is similar to the tri-state logic error function presented in [76], except we explicitly do not reward pixels that are both 0 in the intersection term, as we only want to capture associations between events, and not between pixels without events.

The final cost can be thought of as an analogy to the intersection over union cost:

$$C_{IoU}(x,y,d) = -\frac{C_I(x,y,d)}{C_U(x,y,d)} \tag{5.8}$$

Minimizing this final cost will implicitly maximize the similarity between the two windows, while minimizing the blurring in each. By computing this cost function at every pixel and disparity, we generate a cost volume, where each element (x,y,d) in the volume contains the cost of pixel (x,y) being at disparity d.

5.3.4 Disparity Estimation

Given the cost volume, the fastest way to obtain the estimate for the true disparity at each pixel is to compute the argmax across the disparity dimension of the cost volume, in a winner takes all fashion:

$$\hat{d}(x,y) = \arg\min_d \; C_{IoU}(x,y,d) \tag{5.9}$$

However, we can also apply any traditional optimization method for stereo disparity estimation over the cost volume, such as semi-global matching [38] or belief propagation [14].

5.3.5 Outlier Rejection

While the cost function in (5.8) was relatively robust in our experiments, there were still some regions of the image where it was unable to resolve the correct disparity, which we need to remove from the final output. In particular, we found that pixels with a low final IoU cost typically corresponded to pure noise in the image, where the number of intersection matches was low compared to the number of events in the window. Therefore, any disparities with C_{IoU} less than a parameter ϵ_c are considered outliers. In addition, windows with a low number of events do not provide enough support to find a meaningful match, and so we consider outliers any disparities with C_U less than $\epsilon_n \times \|W\|$, where ϵ_n is a parameter and $\|W\|$ is the number of pixels in the spatial neighborhood.

5.4 Implementation Details

In our experiments, unless otherwise stated, we use a disparity range ranging from 0 to 31 pixels, and a square window with side length of 24 pixels. For outlier rejection, ϵ_c and ϵ_n were both set to 0.1. At each time step, a constant number of events is passed to the algorithm. For our experiments, we used 15,000 events.

As every step of the algorithm is vectorizable with matrix notation, the algorithm was efficiently implemented on GPU in Tensorflow. In particular, (5.2) and (5.3) are implemented as a matrix operations, (5.4) and (5.5) are performed using scatter_nd, and the costs in (5.6) and (5.7) are computed by computing the costs for each pixel at each disparity, and applying two 1D depthwise convolutions with a kernel of ones of the same length as the window size (one along the rows, one along the columns).

With all operations fully vectorized, the algorithm takes 40ms to run on a laptop NVIDIA 960M GPU, including transfer time to the GPU. With further optimizations and an implementation in raw CUDA or OpenCL, we expect this time to reduce further. This corresponds to a runtime of around 2.7μs per event, compared to the 0.65-2ms reported in [154]. However, it should be noted that the competing methods were implemented in MATLAB on CPU, and would almost certainly see speed improvements if ported to other languages/devices. In addition, our method is relatively insensitive to the number of events, as a large proportion of the run time (~40%) is consumed in the sliding window cost. For example, processing a window of 30,000 events takes 46ms to run, corresponding to a runtime of 1.53μs per event.

5.5 Experiments

5.5.1 Data

We evaluated our algorithm on the Multi Vehicle Stereo Event Camera (MVSEC) dataset [170]. MVSEC provides data captured from a stereo event camera pair, along with grayscale images and ground truth depth and pose of the cameras. We tested our method on the indoor_flying sequences, and evaluated against the provided ground truth depth maps. These sequences were generated from a stereo event camera pair mounted on a hexacopter, and flown in an indoor environment, with ground truth generated from lidar measurements. In particular, we used the following depth map frames (zero index) from these indoor_ flying sequences for evaluation: indoor_flying1:

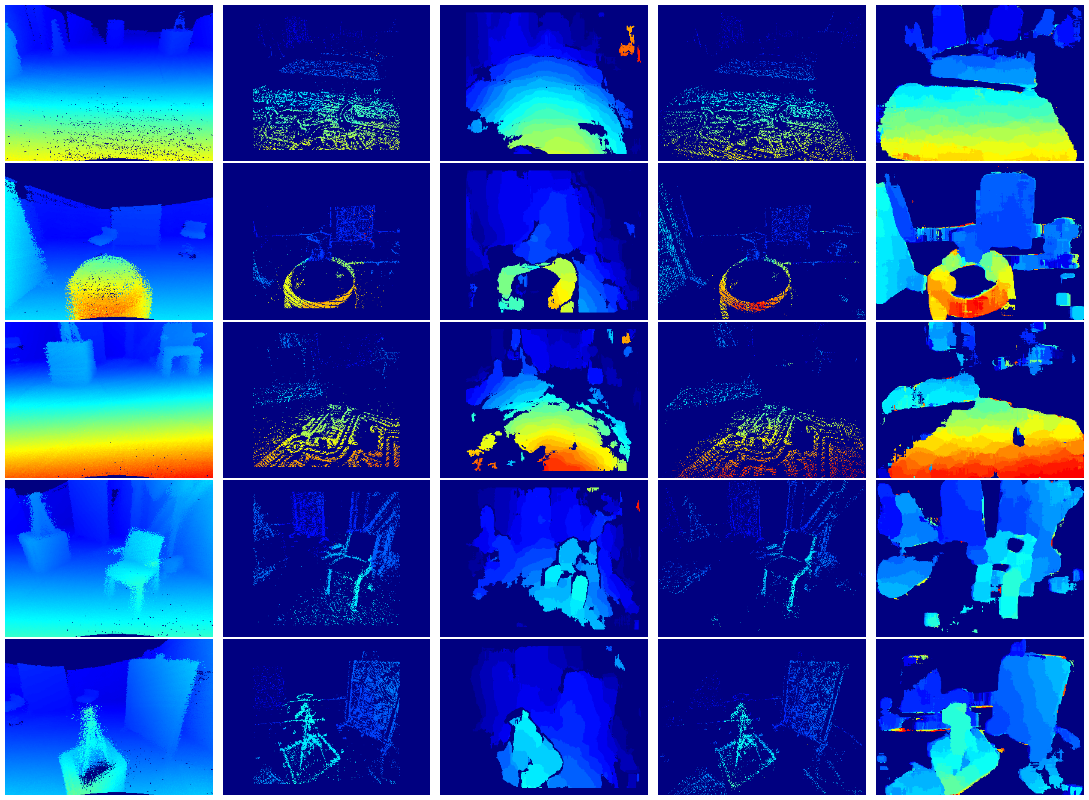


Figure 23: Sample outputs from TSES (our method), compared against CopNet and block matching. Ground truth from MVSEC. Pixels without disparities are dark blue. Note that the border of the CopNet and block matching results are empty due to the window size. Quantitative results were only computed over points with disparities. Best viewed in color.

140-1200, indoor_flying2: 120-1420, indoor_flying3: 73-1616. These frames were selected to exclude the takeoff and landing frames where the ground is closer than our selected maximum disparity.

The driving sequences were not included as the majority of the points in those sequences were beyond the depth resolved by a disparity of 1, and so a sub-pixel disparity estimator would be needed to achieve accurate results. In addition, we do not include results from indoor_flying4, as the majority of events are closer than the maximum disparity of 31, and are also generated by the low-texture floor, on which we could not generate reasonable results with any of the methods.

	Mean Disp. Error (pix)			Mean Depth Error (m)			% Disp. Err < 1		
	IF1	IF2	IF3	IF1	IF2	IF3	IF1	IF2	IF3
TSES	0.89	1.98	0.88	0.36	0.44	0.36	**82.3**	**70.1**	**82.3**
CopNet	1.03	1.54	1.01	0.61	1.00	0.64	70.4	52.8	70.6
BM	**0.73**	**1.02**	**0.82**	**0.23**	**0.21**	**0.27**	79.5	65.2	74.3
SGBM	1.96	3.06	1.86	0.38	0.38	0.41	69.9	56.8	66.7
Algorithm Ablation									
T-S	1.30	2.54	1.39	0.50	0.58	0.57	77.3	64.9	76.7
I-S	1.71	3.59	1.99	0.67	0.99	0.77	74.2	60.7	72.5
IoU-NS	1.43	2.29	1.42	0.52	0.47	0.53	67.8	59.0	68.3
T-NS	1.85	2.78	1.84	0.76	0.66	0.78	64.2	55.6	64.0
I-NS	2.21	3.20	2.12	0.80	0.80	0.78	61.6	53.5	62.7
TSES w/ outliers	1.87	2.83	1.73	1.28	1.18	1.15	74.3	64.3	75.2
Velocity Noise Ablation									
0%	**0.89**	1.98	**0.88**	**0.36**	**0.44**	**0.36**	**82.3**	70.1	82.3
5%	0.90	**1.97**	**0.88**	**0.36**	0.45	**0.36**	82.0	**70.5**	**82.4**
10%	0.91	1.98	**0.88**	0.37	0.45	**0.36**	81.6	70.1	82.3
20%	0.96	2.04	0.92	0.38	0.46	0.38	80.4	68.6	81.3
50%	1.21	2.44	1.23	0.47	0.58	0.51	74.5	61.5	74.5
100%	1.97	3.47	2.17	0.83	0.92	1.03	61.8	48.5	59.1
Window Size Ablation									
8 pix.	2.40	3.83	2.52	0.78	0.87	0.86	65.4	55.3	63.8
16 pix.	1.10	2.29	1.13	0.43	0.51	0.45	80.3	69.2	79.8
24 pix.	0.89	1.98	0.88	0.36	0.44	0.36	**82.3**	**70.1**	**82.3**
32 pix.	**0.86**	**1.97**	**0.84**	**0.34**	**0.43**	0.34	81.2	66.7	81.4
40 pix.	0.89	2.05	0.91	**0.34**	0.44	**0.33**	78.6	61.9	77.9

Table 4: Quantitative results from testing on the indoor_flying (IF) sequences of TSES (our method) and CopNet, along with ablation studies. Prefixes for the algorithm ablation are: IoU - Intersection over Union cost (5.8), I - Intersection cost (5.7), T - Time cost (5.10). Suffixes are with (S) and without (NS) time synchronization (5.3). Velocity noise was added to the linear and angular velocities separately, as zero mean Gaussian noise with variance equal to a percentage of the norm of each velocity.

While our method generates disparity values whenever there are any events inside the spatial window, we report our results based on disparities on pixels where events appeared, in order to provide a fair comparison with other works.

We used the camera velocities provided in the dataset from [173], which were generated by linear interpolation of the lidar odometry poses provided from MVSEC, and are provided in addition to ground truth optical flow for the sequences in the dataset.

5.5.2 Comparisons

For comparison, we have implemented the CopNet method by Piatkowska et al. [115], and we include their results on the same dataset, using their provided parameters. For these experiments, we have used an α value of 1 (the scaling term in the matching cost, equation (3) in their paper), as the original paper stated a value of 0, which would result in a constant cost. In addition, we compare against block matching and semi-global block matching methods from OpenCV[1], applied to the grayscale frames from the DAVIS camera. Note that the grayscale frames are not time synchronized, and the time offset between the left and right frames is 4ms, 14ms and 14ms for indoor_flying 1, 2 and 3, respectively. However, we were still able to achieve reasonable performance. The quantitative results of these comparisons can be found in Tab. 4.

In addition, we attempted an implementation of the belief propagation based work by Xie et al. [154], but were unable to obtain reasonable results over this dataset, which is significantly more complex than those evaluated in the original work, consisting of a few objects moving in the scene. We believe that this is because their matching cost ($D(d_p)$) attempts to match individual events, without using the spatial neighborhood around the event. In our experiments, this matching cost failed to identify the correct disparity over the majority of the image, which we believe led the belief propagation to output incorrect results.

[1]`https://docs.opencv.org/3.4/d2/d6e/classcv_1_1StereoMatcher.html`

5.5.3 Ablation Studies

In addition to the comparisons, we performed a number of ablation studies over the parameters of the algorithm. All results can be found in Tab. 4.

Algorithm Ablation

To test the effect of the time synchronized event disparity volumes, we performed additional experiments where the raw event positions were passed directly into (5.4) and (5.5) (i.e. by setting $(x'(d)_i, y'(d)_i) = (x_i, y_i)$). Experiments with and without time synchronization are denoted with the suffix -S and -NS, respectively.

To test the IoU cost, we tested with only the intersection cost (prefix I), as well as using the cost function from [115] (prefix T), which is defined as:

$$C_T(x,y,d) = \sum_{x^*,y^* \in W(x,y,d)} \frac{1}{(\alpha \cdot |I_L^t(x^*,y^*,d) - I_R^t(x^*,y^*,d)| + 1) \cdot C_U(x^*,y^*,d)} \tag{5.10}$$

where α is set to 1, the event images I^t now represent the timestamp of the last event to arrive at each pixel and disparity:

$$I_L^t(x,y,d) = \max_{t_i} \{t_i | (x_i'(d), y_i'(d)) = (x,y)\} \tag{5.11}$$

$$I_R^t(x,y,d) = \max_{t_i} \{t_i | (x_i'(d) + d, y_i'(d)) = (x,y)\} \tag{5.12}$$

and we use our union cost in place of the number of events in the left window. We also tested with the inverse of the union cost, but this did not produce any reasonable results.

We also provide results of our full method, without the outlier rejection step.

Velocity Noise Ablation

In practice, it is difficult to estimate the cameras' velocity with the same accuracy as the ground truth. To test the effect of noise on the velocity estimate, we perform additional experiments where we add zero mean Gaussian noise to the linear and angular velocities. The variance of the noise is set to a given percentage of the norm of the linear and angular velocities, separately.

Window Size Ablation

We also tested the effects of the window size on the performance of our method over a range of window sizes.

5.5.4 Results and Discussion

Comparisons

We present some qualitative results in Fig. 23, comparing our method to CopNet, block matching and ground truth. While both sets of results look visually reasonable, we can see that our method suffers less from foreground fattening [47] (e.g. the chair in the fourth row). Our method does, however, tend to produce erroneous results on the edges of images in the dense disparity image, but these correspond to pixels without any events, and are thresholded away in the sparse disparity image.

In addition, we provide quantitative results in Tab. 4, where we can see that our full method outperforms CopNet across almost every measure, as well as the other methods in the ablation study. In particular, while the disparity errors are similar, CopNet performs significantly worse in depth overall. Upon examining the results. We found that this error from CopNet was largely due to the fact that the method had over-smoothed the disparity output. This was mostly due to the window size used, which is relatively large (39x39). This oversmoothing tends to pull far away points closer (overestimates disparities), which leads to higher depth errors, as they are higher at lower disparity levels. However, we observed that reducing the window sizes resulted in a further reduction in the overall matching accuracy due to increased ambiguity in the matching, as noted by the authors in the original paper [115], so there was no immediate solution for this problem.

The block matching method performed better in terms of mean errors across all three sequences, although the mean disparity errors for sequences 1 and 3 are both less than 1 pixel, which is within the range of the discretization error. In addition, our method has a higher percentage of points with disparity error <1 across all sequences.

We were unable to achieve comparable performance from semi-global block matching, which tended to over smooth incorrect regions in the image.

Algorithm Ablations

From the ablation study, we can see that removing each component of the method tends to result in a corresponding decrease in accuracy, with the time synchronization always resulting in better results. In addition, the addition of the union cost to the overall cost provides a significant improvement in accuracy over the intersection cost, which is a pure similarity measure.

Furthermore, we can see that our IoU cost outperforms the timestamp based cost in both situations, suggesting that it may be a better alternative for more complex methods, even without the time synchronization. When the proposed time synchronization is applied at the correct disparity, older timestamps are mapped to later timestamps from the same point in the image. As the time cost operates on an image that only keeps the latest timestamp, this results in images with timestamps that are very similar (all later events), which do not provide much discriminative power. Future work could consider all of the events that map to a pixel, but this requires a new cost function.

Finally, the results without outlier rejection have significantly higher mean disparity errors, suggesting that a large number of outliers were rejected by our method, while from the % disparity error < 1 results, we can see that only <10% of the points were rejected.

Velocity Noise Ablation

The velocity noise ablation results show stable errors up to noise with variance up to around 20% of the velocity norm. We believe that a conventional state estimation pipeline for event cameras should be able to reliably estimate the camera velocity within these error bounds.

Window Size Ablation

We found that window sizes between 24 and 40 pixels achieved the best results. However, larger window sizes increase the amount of foreground fattening, as well as the runtime of the algorithm. Therefore, we recommend a window size of 24 pixels for these test cases. In terms of run time, the algorithm took 33ms, 40ms and 50ms to run for window sizes of 16, 24 and 32 pixels, respectively. Similarly, the runtime was 25ms and 60ms for disparity ranges of 16 and 48, with a window size of 24 pixels.

5.6 Conclusions

In this chapter, we have proposed a novel method for stereo event disparity matching which uses the concept of motion blur to synchronize the event streams in time. We show that our method, consisting of a simple temporal interpolation of the events, along with a lightweight matching cost, is able to outperform state of the art methods which perform expensive regularizations on top of the disparity map. In addition, as our disparity results are at a single time, analogous to an image frame, they can be directly passed into any frame based architecture such as a state estimator, as compared to an asynchronous disparity stream. We envision that this method will be coupled with a method for estimating camera velocity, such as a visual odometry algorithm, for real time performance.

Chapter 6

EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras

6.1 Introduction

For traditional image-based methods, deep learning has helped the computer vision community achieve new levels of performance while avoiding having to explicitly model the entire problem. However, these techniques have yet to see the same level of adoption and success for event-based cameras. One reason for this is the asynchronous output of the event-based camera, which does not easily fit into the synchronous, frame-based inputs expected by image-based paradigms. Another reason is the lack of labeled training data necessary for supervised training methods. In this chapter, we propose two main contributions to resolve these issues.

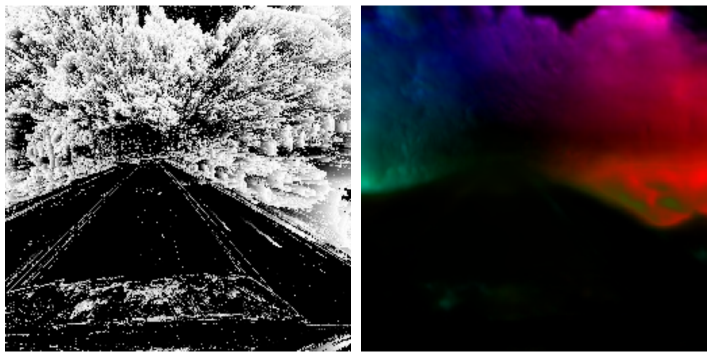


Figure 24: Left: Event input to the network visualizing the last two channels (latest timestamps). Right: Predicted flow, colored by direction. Best viewed in color.

First, we propose a novel image-based representation of an event stream, which fits into any standard image-based neural network architecture. The event stream is summarized by an image with channels representing the number of events and the latest timestamp at each polarity at each pixel. This compact representation preserves the spatial relationships between events, while maintaining the most recent temporal information at each pixel and providing a fixed number of channels for any event stream.

Second, we present a self-supervised learning method for optical flow estimation given only a set of events and the corresponding grayscale images generated from the same camera. The self-supervised loss is modeled after frame based self-supervised flow networks such as Yu et al. [158] and Meister et al. [93], where a photometric loss is used as a supervisory signal in place of direct supervision. As a result, the network can be trained using only data captured directly from an event camera that also generates frame based images, such as the Dynamic and Active-pixel Vision (DAVIS) Sensor developed by Brandli et al. [18], circumventing the need for expensive labeling of data.

These event images combined with the self-supervised loss are sufficient for the network to learn to predict accurate optical flow from events alone. For evaluation, we generate a new event camera optical flow dataset, using the ground truth depths and poses in the Multi Vehicle Event Camera Dataset (Chapter 4). We show that our method is competitive on this dataset with UnFlow by Meister et al. [93], an image-based self supervised network trained on KITTI, and fine tuned on event camera frames, as well as standard non-learning based optical flow methods.

In summary, our main contributions in this work are:

- We introduce a novel method for learning optical flow using events as inputs only, without any supervision from ground-truth flow.
- Our CNN architecture uses a self-supervised photoconsistency loss from low resolution intensity images used in training only.
- We present a novel event-based optical flow dataset with ground truth optical flow, on which we evaluate our method against a state of the art frame based method.

6.2 Related Work

6.2.1 Event-based Optical Flow

There have been several works that attempt to take advantage of the high temporal resolution of the event camera to estimate accurate optical flow. Benosman et al. [12] model a given patch moving in the spatial temporal domain as a plane, and estimate optical flow as the slope of this plane. This work is extended in Benosman et al. [13] by adding an iterative outlier rejection scheme to remove events significantly far from the plane, and in Barranco et al. [7] by combining the estimated flow with flow from traditional images. Brosch et al. [19] present an analogy of Lucas and Kanade [89] using the events to approximate the spatial image gradient, while Orchard and Etienne-Cummings [110] use a spiking neural network to estimate flow, and Liu and Delbruck [88] estimate sparse flow using an adaptive block matching algorithm. In other works, Bardow et al. [6] present the optical flow estimation problem jointly with image reconstruction, and solve the joint problem using convex optimization methods, while Zhu et al. [167] present an expectation-maximization based approach to estimate flow in a local patch. A number of these methods have been evaluated in Rueckauer and Delbruck [130] against relatively simple scenes with limited translation and rotation, with limited results, with ground truth optical flow estimated from a gyroscope. Similarly, Barranco et al. [8] provide a dataset with optical flow generated from a known motion combined with depths from a RGB-D sensor.

6.2.2 Event-based Deep Learning

One of the main challenges for supervised learning for events is the lack of labeled data. As a result, many of the early works on learning with event-based data, such as Ghosh et al. [50] and Moeys et al. [97], rely on small, hand collected datasets.

To address this, recent works have attempted to collect new datasets of event camera data. Mueggler et al. [103], provide handheld sequences with ground truth camera pose, which Nguyen et al. [106] use to train a LSTM network to predict camera pose. In addition, Zhu et al. [172] provide flying, driving and handheld sequences with ground truth camera pose and depth maps, and Binas et al. [16] provide long driving sequences with ground truth measurements from the vehicle such as steering angle and GPS position.

Another approach has been to generate event based equivalents of existing image based datasets by recording images from these datasets from an event based camera (Orchard et al. [111], Hu et al. [65]).

Recently, there have also been implementations of neural networks on spiking neuromorphic processors, such as in Amir et al. [2], where a network is adapted to the TrueNorth chip to perform gesture recognition.

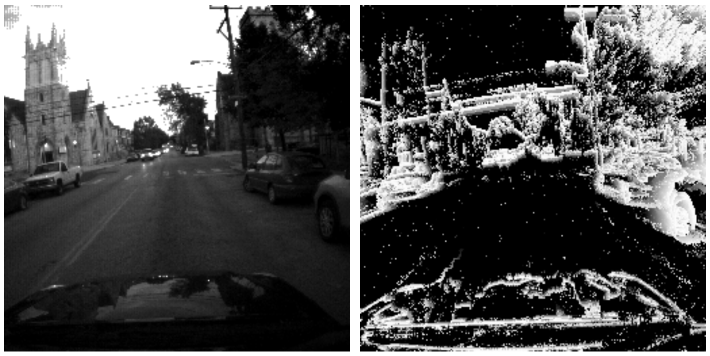


Figure 25: Example of a timestamp image. Left: Grayscale output. Right: Timestamp image, where each pixel represents the timestamp of the most recent event. Brighter is more recent.

6.2.3 Self-supervised Optical Flow

Self-supervised, or unsupervised, methods have shown great promise in training networks to solve many challenging 3D perception problems. Yu et al. [158] and Ren et al. [127] train an optical flow prediction network using the traditional brightness constancy and smoothness constraints developed in optimization based methods such as the Lucas Kanade method Lucas and Kanade [89]. Zhu et al. [176] combine this self-supervised loss with supervision from an optimization based flow estimate as a proxy for ground truth supervision, while Meister et al. [93] extend the loss with occlusion masks and a second order smoothness term, and Lai et al. [82] introduce an adversarial loss on top of the photometric error.

6.3 Method

In this section, we describe our approach in detail. In Sec. 6.3.1, we describe our event representation, which is an analogy to an event image. In Sec. 6.3.2, we describe the self-supervised loss used to provide a supervisory signal using only the gray scale images captured before and after each time window, and in Sec. 6.3.3, we describe

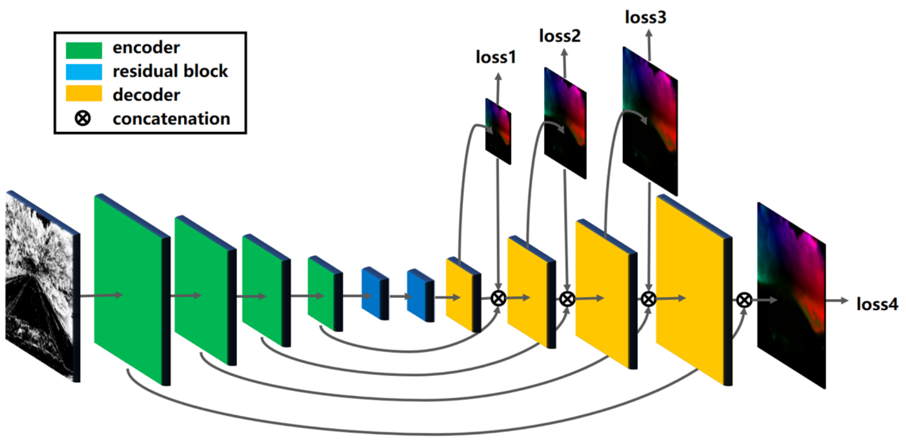


Figure 26: EV-FlowNet architecture. The event input is downsampled through four encoder (strided convolution) layers, before being passed through two residual block layers. The activations are then passed through four decoder (upsample convolution) layers, with skip connections to the corresponding encoder layer. In addition, each set of decoder activations is passed through another depthwise convolution layer to generate a flow prediction at its resolution. A loss is applied to this flow prediction, and the prediction is also concatenated to the decoder activations. Best viewed in color.

the architecture of our network, which takes as input the event image and outputs a pixel-wise optical flow. Note that, throughout this paper, we refer to optical flow as the displacement of each pixel within a given time window.

6.3.1 Event Representation

An event-based camera tracks changes in the log intensity of an image, and returns an event whenever the log intensity changes over a set threshold θ:

$$\|\log(I_{t+1}(\mathbf{x})) - \log(I_t(\mathbf{x}))\| \geq \theta \tag{6.1}$$

Each event contains the pixel location of the change, timestamp of the event and polarity:

$$e = \left\{ \mathbf{x}, \quad t, \quad p \right\} \tag{6.2}$$

Because of the asynchronous nature of the events, it is not immediately clear what representation of the events should be used in the standard convolutional neural network architecture. Most modern network architectures expect image-like inputs, with a fixed, relatively low, number of channels (recurrent networks excluded) and spatial correlations between neighboring pixels. Therefore, a good representation is key to fully take advantage of existing networks while summarizing the necessary information from the event stream.

Perhaps the most complete representation that preserves all of the information in each event would be to represent the events as a $n \times 4$ matrix, where each column contains the information of a single event. However, this does not directly encode the spatial relationships between events that is typically exploited by convolutions over images.

In this work, we chose to instead use a representation of the events in image form. The input to the network is a 4 channel image with the same resolution as the camera.

The first two channels encode the number of positive and negative events that have occurred at each pixel, respectively. This counting of events is a common method for visualizing the event stream, and has been shown in Nguyen et al. [106] to be informative in a learning based framework to regress 6dof pose.

However, the number of events alone discards valuable information in the timestamps that encode information about the motion in the image. Incorporating timestamps in image form is a challenging task. One possible solution would be to have k channels, where k is the most events in any pixel in the image, and stack all incoming timestamps. However, this would result in a large increase in the dimensionality of the input. Instead, we encode the pixels in the last two channels as the timestamp of the most recent positive and negative event at that pixel, respectively. This is similar to the "Event-based Time Surfaces" used in Lagorce et al. [81] and the "timestamp images" used in Park et al. [112]. An example of this kind of image can be found in Fig. 25, where we can see that the flow is evident by following the gradient in the image, particularly for closer (faster moving) objects. While this representation inherently discards all of the timestamps but the most recent at each pixel, we have observed that this representation is sufficient for the network to estimate the correct flow in most regions. One deficiency of this representation is that areas with very dense events and large motion will have all pixels overridden by very recent events with very similar timestamps. However, this problem can be avoided by choosing smaller time windows, thereby reducing the magnitude of the motion.

In addition, we normalize the timestamp images by the size of the time window for the image, so that the maximum value in the last two channels is 1. This has the effect of both scaling the timestamps to be on the same order of magnitude as the event counts, and ensuring that fast motions with a small time window and slow motions with a large time window that generate similar displacements have similar inputs to the network.

6.3.2 Self-Supervised Loss

Due to the fact that there is a relatively small amount of labeled data for event based cameras as compared to traditional cameras, it is difficult to generate a sufficient dataset for a supervised learning method. Instead, we utilize the fact that the DAVIS camera generates synchronized events and grayscale images to perform self-supervised learning using the grayscale images in the loss. At training time, the network is provided with the event timestamp images, as well as a pair of grayscale images, occurring immediately before and after the event time window. Only the event timestamp images are passed into the network, which predicts a per pixel flow. The grayscale images are then used to apply a loss over the predicted flow in a self-supervised manner.

The overall loss function used follows traditional variational methods for estimating optical flow, and consists of a photometric and a smoothness loss.

To compute the photometric loss, the flow is used to warp the second image to the first image using bilinear sampling, as described in Yu et al. [158]. The photometric loss, then, aims to minimize the difference in intensity between the warped second image and the first image:

$$\begin{aligned}\ell_{\text{photometric}}(u, v; I_t, I_{t+1}) = \\ \sum_{x,y} \rho(I_t(x, y) - I_{t+1}(x + u(x, y), y + v(x, y)))\end{aligned} \tag{6.3}$$

where ρ is the Charbonnier loss function, a common loss in the optical flow literature used for outlier rejection (Sun et al. [138]):

$$\rho(x) = (x^2 + \epsilon^2)^\alpha \tag{6.4}$$

As we are using frame based images for supervision, this method is susceptible to image-based issues such as the aperture problem. Thus, we follow the other works in the frame based domain, and apply a regularizer in the form of a smoothness loss. The smoothness loss aims to regularize the output flow by minimizing the difference in flow between neighboring pixels horizontally, vertically and diagonally.

$$\begin{aligned}&\ell_{\text{smoothness}}(u,v) = \\ &\sum_{x,y}\sum_{i,j\in\mathcal{N}(x,y)} \rho(u(x,y)-u(i,j))+\rho(v(x,y)-v(i,j))\end{aligned} \tag{6.5}$$

where $\mathcal{N}$ is the set of neighbors around (x, y).

The total loss is the weighted sum of the photometric and smoothness losses:

$$L_{\text{total}} = \ell_{\text{photometric}} + \lambda\ell_{\text{smoothness}} \tag{6.6}$$

6.3.3 Network Architecture

The EV-FlowNet architecture very closely resembles the encoder-decoder networks such as the stacked hourglass (Newell et al. [105]) and the U-Net (Ronneberger et al. [129]), and is illustrated in Fig. 26. The input event image is passed through 4 strided convolution layers, with output channels doubling each time. The resulting activations are passed through 2 residual blocks, and then four upsample convolution layers, where the activations are upsampled using nearest neighbor sampling and then convolved, to obtain a final flow estimate. At each upsample convolution layer, there is also a skip connection from the corresponding strided convolution layer, as well as another convolution layer to produce an intermediate, lower resolution, flow estimate, which is concatenated with the activations from the upsample convolution. The loss in (6.6) is then applied to each intermediate flow by downsampling the grayscale images. The tanh function is used as the activation function for all of the flow predictions.

6.4 Optical Flow Dataset

For ground truth evaluation only, we generated a novel dataset for ground truth optical flow using the data provided in the Multi-Vehicle Stereo Event Camera dataset (MVSEC) by Zhu et al. [172]. The dataset contains stereo event camera data in a number of flying, driving and handheld scenes. In addition, the dataset provides ground truth poses and depths maps for each event camera, which we have used to generate reference ground truth optical flow.

From the pose (consisting of rotation R and translation $\mathbf{p}$) of the camera at time t_0 and t_1, we make a linear velocity assumption, and estimate velocity and angular velocity using numerical differentiation:

$$\mathbf{v} = \frac{(\mathbf{p}(t_1) - \mathbf{p}(t_0))}{dt} \tag{6.7}$$

$$\omega^\wedge = \frac{\operatorname{logm}\left(R_{t_0}^T R_{t_1}\right)}{dt} \tag{6.8}$$

where logm is the matrix logarithm, and $\omega^\wedge$ converts the vector ω into the corresponding skew symmetric matrix:

$$\omega^\wedge = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{6.9}$$

A central moving average filter is applied to the estimated velocities to reduce noise. We then use these velocities to estimate the motion field, given the ground truth depths, Z, at each undistorted pixel position:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{bmatrix} -\frac{1}{Z} & 0 & -\frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & -\frac{1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{bmatrix} \begin{pmatrix} \mathbf{v} \\ \omega \end{pmatrix} \tag{6.10}$$

Finally, we scale the motion field by the time window between each pair of images dt, and use the resulting displacement as an approximation to the true optical flow for each pixel. To apply the ground truth to the distorted images, we shift the undistorted pixels by the flow, and apply distortion to the shifted pixels. The distorted flow is, then, the displacement from the original distorted position to the shifted distorted position.

In total, we have ground truth optical flow for the indoor_flying, outdoor_day and outdoor_night sequences. In addition to using the indoor_flying and outdoor_day ground truth sets for evaluation, we will also release all sequences as a dataset.

6.5 Empirical Evaluation

6.5.1 Training Details

We trained two networks separately on the two outdoor_day sequences from MVSEC. outdoor_day1 contains roughly 12000 images, and outdoor_day2 contains roughly 26000 images. The images are captured from driving in an industrial complex and public roads, respectively, where the two scenes are visually very different. The motions include mostly straights and turns, with occasional independently moving objects such as other cars and pedestrians. The input images are cropped to 256x256, the number of output channels at the first encoder layer is 64 and the number of output channels in each residual block is 512.

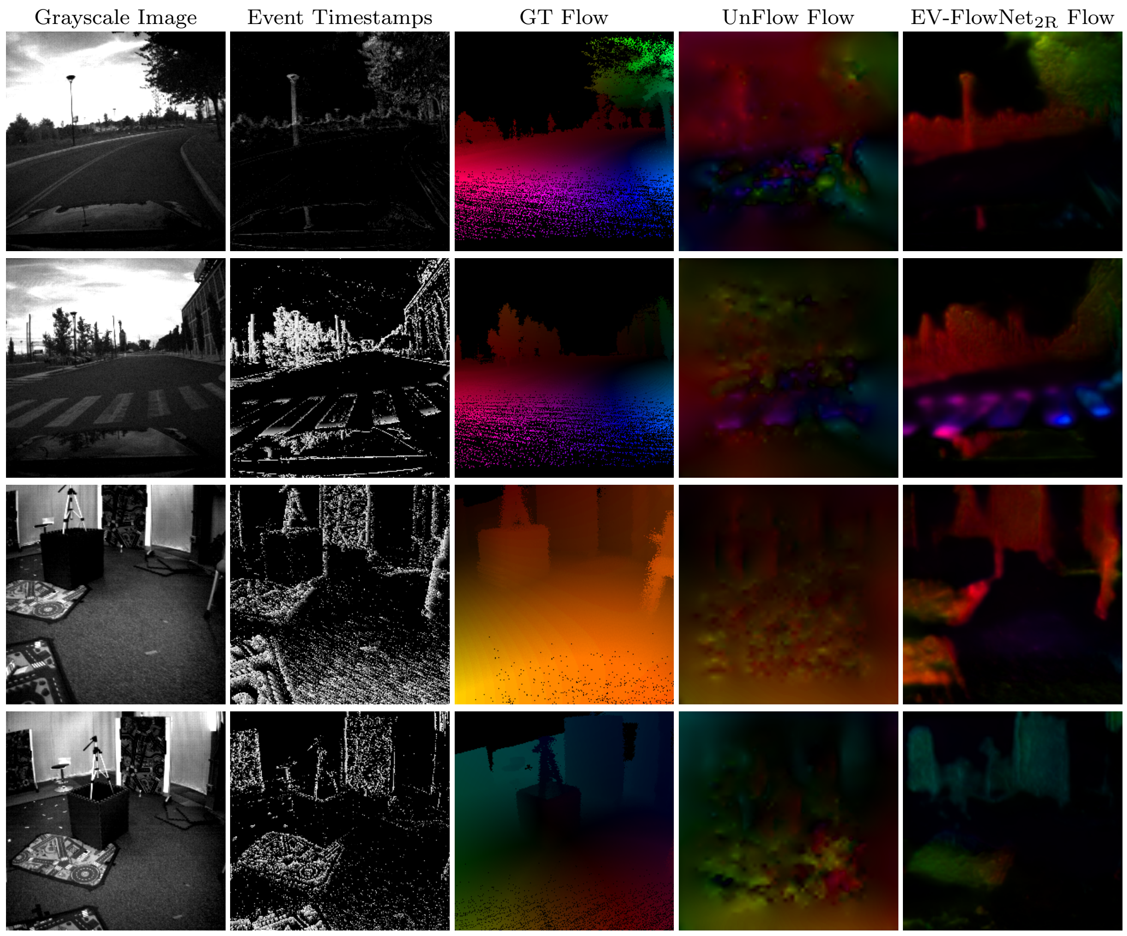


Figure 27: Qualitative results from evaluation. Examples were collected from outdoor day1, outdoor day1, indoor flying1 and indoor flying2, top to bottom. Best viewed in color.

To increase the variation in the magnitude of the optical flow seen at training, we randomly select images up to k images apart in time, and all of the events that occurred between those images. In our experiments, $k \in [2, 4, 6, 8, 10, 12]$. In addition, we randomly flip the inputs horizontally, and randomly crop them to achieve the desired resolution.

The weight on the smoothness loss (6.6), λ, is set to 0.5. Each of the intermediate losses is weighted equally in the final loss. For the Charbonnier loss (6.4), α was set to be 0.45 and ϵ was set to be 1e-3. The Adam optimizer is used, with learning rate initialized at 1e-5, and exponentially decayed every 4 epochs by 0.8. The model is trained for 300,000 iterations, and takes around 12 hours to train on a 16GB NVIDIA Tesla V100.

6.5.2 Ablation Studies

In addition to the described architecture (denoted EV-FlowNet$_{\mathrm{2R}}$), we also train three other networks to test the effects of varying the input to the network, as well as increasing the capacity of the network.

To test the contribution of each of the channels in the input, we train two additional networks, one with only the event counts (first two channels) as input (denoted EV-FlowNet$_{\mathrm{C}}$), and one with only the event timestamps (last two channels) as input (denoted EV-FlowNet$_{\mathrm{R}}$).

In addition, we tested different network capacities by training a larger model with 4 residual blocks (denoted EV-FlowNet$_{\mathrm{4R}}$). A single forward pass takes, on average, 40ms for the smaller network, and 48ms for the larger network, when run on a NVIDIA GeForce GTX 1050, a laptop grade GPU.

6.5.3 Comparisons

To compare our results with other existing methods, we tested implementations of Event-based Visual Flow by Benosman et al. [13], an optimization based method that works on events, and UnFlow by Meister et al. [93], a self supervised method that works on traditional frames.

As there is no open source code by the authors of Event-based Visual Flow, we designed an implementation around the method described in Rueckauer and Delbruck

dt=1 frame	outdoor driving		indoor flying1		indoor flying2		indoor flying3	
	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier
UnFlow	0.97	1.6	**0.50**	**0.1**	**0.70**	**1.0**	**0.55**	**0.0**
EV-FlowNet$_{\mathrm{C}}$	**0.49**	**0.2**	1.30	6.8	2.34	25.9	2.06	22.2
EV-FlowNet$_{\mathrm{T}}$	0.52	**0.2**	1.20	4.5	2.15	22.6	1.91	19.8
EV-FlowNet$_{\mathrm{2R}}$	**0.49**	**0.2**	1.03	2.2	1.72	15.1	1.53	11.9
EV-FlowNet$_{\mathrm{4R}}$	**0.49**	**0.2**	1.14	3.5	2.10	21.0	1.85	18.8
dt=4 frames	outdoor driving		indoor flying1		indoor flying2		indoor flying3	
	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier
UnFlow	2.95	40.0	3.81	56.1	6.22	79.5	**1.96**	**18.2**
EV-FlowNet$_{\mathrm{C}}$	1.41	10.8	3.22	41.4	5.30	60.1	4.68	57.0
EV-FlowNet$_{\mathrm{T}}$	1.34	8.4	2.53	33.7	4.40	51.9	3.91	47.1
EV-FlowNet$_{\mathrm{2R}}$	**1.23**	**7.3**	**2.25**	**24.7**	**4.05**	**45.3**	3.45	39.7
EV-FlowNet$_{\mathrm{4R}}$	1.33	9.4	2.75	33.5	4.82	53.3	4.30	47.8

Table 5: Quantitative evaluation of each model on the MVSEC optical flow ground truth. Average end-point error (AEE) and percentage of pixels with EE above 3 and 5% of the magnitude of the flow vector(% Outlier) are presented for each method (lower is better for both), with evaluation run with image pairs 1 frame apart (top) and 4 frames apart (bottom). The EV-FlowNet methods are: Counts only (EV-FlowNet$_{\mathrm{c}}$), Timestamps only (EV-FlowNet$_{\mathrm{T}}$), 2 Res. blocks (EV-FlowNet$_{\mathrm{2R}}$) and 4 Res. blocks (EV-FlowNet$_{\mathrm{4R}}$).

[130]. In particular, we implemented the robust Local Plane Fit algorithm, with a spatial window of 5x5 pixels, vanishing gradient threshold th3 of 1e-3, and outlier distance threshold of 1e-2. However, we were unable to achieve any reasonable results on the datasets, with only very few points returning valid flow values ($< 5\%$), and none of the valid flow values being visually correct. For validation, we also tested the open source MATLAB code provided by the authors of Mueggler et al. [103], where we received similar results. As a result, we believe that the method was unable to generalize to the natural scenes in the test set, and so did not include the results in this paper.

For UnFlow, we used the unsupervised model trained on KITTI raw, and fine tuned on outdoor_day2. This model was able to produce reasonable results on the testing sets, and we include the results in the quantitative evaluation in Tab. 5.

6.5.4 Test Sequences

For comparison against UnFlow, we evaluated 800 frames from the outdoor_day1 sequence as well as sequences 1 to 3 from indoor_flying. For the event input, we used all of the events that occurred in between the two input frames.

The outdoor_day1 sequence spans between 222.4s and 240.4s. This section was chosen as the grayscale images were consistently bright, and there is minimal shaking of the camera (the provided poses are smoothed and do not capture shaking of the camera if the vehicle hits a bump in the road). In order to avoid conflicts between training and testing data, a model trained only using data from outdoor_day2 was used, which is visually significantly different from outdoor_day1.

The three indoor_flying sequences total roughly 240s, and feature a significantly different indoor scene, containing vertical and backward motions, which were previously unseen in the driving scenes. A model trained on both outdoor_day1 and outdoor_day2 data was used for evaluation on these sequences. We avoided fine tuning on the flying sequences, as the sequences are in one room, and all relatively similar in visual appearance. As a result, it would be very easy for a network to overfit the environment. Sequence 4 was omitted as the majority of the view was just the floor, and so had a relatively small amount of useful data for evaluation.

6.5.5 Metrics

For each method and sequence, we compute the average endpoint error (AEE), defined as as the distance between the endpoints of the predicted and ground truth flow vectors:

$$AEE = \sum_{x,y} \left\| \begin{pmatrix} u(x,y)_{\text{pred}} \\ v(x,y)_{\text{pred}} \end{pmatrix} - \begin{pmatrix} u(x,y)_{\text{gt}} \\ v(x,y)_{\text{gt}} \end{pmatrix} \right\|_2 \tag{6.11}$$

In addition, we follow the KITTI flow 2015 benchmark and report the percentage of points with EE greater than 3 pixels and 5% of the magnitude of the flow vector. Similarly to KITTI, 3 pixels is roughly the maximum error observed when warping the grayscale images according to the ground truth flow, and comparing against the next image.

However, as the input event image is relatively sparse, the network only returns accurate flow on points with events. As a result, we limit the computation of AEE to pixels in which at least one event was observed. For consistency, this is done with a mask applied to the EE for both event-based and frame-based methods. We also mask out any points for which we have no ground truth flow (i.e. regions with no ground truth depth). In practice, this results in the error being computed over 20-30% of the pixels in each image.

In order to vary the magnitude of flow observed for each test, we run two evaluations per sequence: one with input frames and corresponding events that are one frame apart, and one with frames and events four frames apart. We outline the results in Tab. 5.

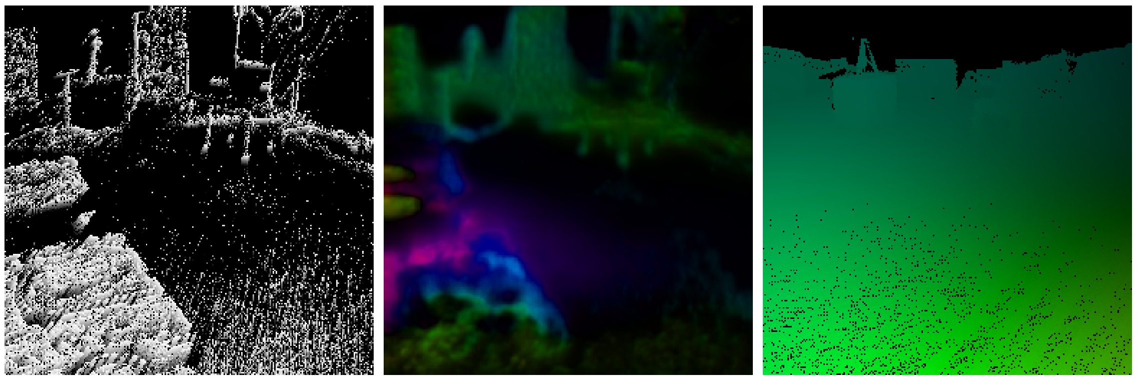


Figure 28: Common failure case, where fast motion causes recent timestamps to overwrite older pixels nearby, resulting in incorrect predictions. Best viewed in color.

6.5.6 Results

Qualitative Results

In addition to the quantitative analysis provided, we provide qualitative results in Fig. 27. In these results, and throughout the test set, the predicted flow always closely follows the ground truth. As the event input is quite sparse, our network tends to predict zero flow in areas without events. This is consistent with the photometric loss, as areas without events are typically low texture areas, where there is little change in intensity within each pixel neighborhood. In practice, the useful flow can be extracted by only using flow predictions at points with events. On the other hand, while UnFlow typically performs reasonably on the high texture regions, the results on low texture regions are very noisy.

Ablation Study Results

From the results of the ablation studies in Tab. 5, EV-FlowNet$_{\mathrm{C}}$ (counts only) performed the worst. This aligns with our intuition, as the only information attainable from the counts is from motion blur effects, which is a weak signal on its own. EV-FlowNet$_{\mathrm{T}}$ (timestamps only) performs better for most tests, as the timestamps carry information about the ordering between neighboring events, as well as the magnitude of the velocity. However, the timestamp only network fails when there is significant noise in the image, or when fast motion results in more recent timestamps covering all of the older ones. This is illustrated in Fig 28, where even the full network struggles to predict the flow in a region dominated by recent timestamps. Overall, the combined models clearly perform better, likely as the event counts carry information about the importance of each pixel. Pixels with few events are likely to be just noise, while pixels with many events are more likely to carry useful information. Somewhat surprisingly, the larger network, EV-FlowNet$_{4R}$ actually performs worse than the smaller one, EV-FlowNet$_{2R}$. A possible explanation is that the larger capacity network learned to overfit the training sets, and so did not generalize as well to the test sets, which were significantly different. For extra validation, both EV-FlowNet$_{2R}$ and EV-FlowNet$_{4R}$ were trained for an additional 200,000 iterations, with no appreciable improvements. It is likely, however, that, given more data, the larger model would perform better.

Comparison Results

From our experiments, we found that the UnFlow network tends to predict roughly correct flows for most inputs, but tends to be very noisy in low texture areas of the image. The sparse nature of the events is a benefit in these regions, as the lack of events there would cause the network to predict no flow, instead of an incorrect output.

In general, EV-FlowNet performed better on the dt=4 tests, while worse on the dt=1 tests (with the exception of outdoor_driving1 and indoor_flying3). We observed that UnFlow typically performed better in situations with very small or very large motion. In these situations, there are either few events as input, or so many events that the image is overriden by recent timestamps. However, this is a problem intrinsic to the testing process, as the time window is defined by the image frame rate. In practice, these problems can be avoided by choosing time windows large enough so that sufficient information is available while avoiding saturating the event image. One possible solution to this would be to have a fixed number of events in the window each time.

6.6 Conclusion

In this chapter, we have presented a novel design for a neural network architecture that is able to accurately predict optical flow from events alone. Due to the method's self-supervised nature, the network can be trained without any manual labeling, simply by recording data from the camera. We show that the predictions generalize beyond hand designed laboratory scenes to natural ones, and that the method is competitive with state of the art frame-based self supervised methods. We hope that this work will provide not only a novel method for flow estimation, but also a paradigm for applying other self-supervised learning methods to event cameras in the future. For future work, we hope to incorporate additional losses that provide supervisory signals from event data alone, to expose the network to scenes that are challenging for traditional frame-based cameras, such as those with high speed motions or challenging lighting.

Chapter 7

Unsupervised Event-based Learning of Optical Flow, Depth and Egomotion

7.1 Introduction

In the previous chapter, as well as the work by Ye et al. [156], methods have been presented which train neural networks to learn to estimate motion in a self and unsupervised manner. These networks abstract away the difficult problem of modeling and algorithm development. However, both works still rely on photoconsistency based principles, applied to the grayscale image and an event image respectively, and, as a result, the former work relies on the presence of grayscale images, while the latter's photoconsistency assumption may not hold valid in very blurry scenes. In addition, both works take inputs that attempt to summarize the event data, and as a result lose temporal information.

In this chapter, we resolve these deficiencies by proposing a novel input representation that captures the full spatiotemporal distribution of the events, and a novel set of unsupervised loss functions that allows for efficient learning of motion information from only the event stream. Our input representation, a discretized event volume, discretizes the time domain, and then accumulates events in a linearly weighted fashion similar to interpolation. This representation encodes the distribution of all of the events within the spatiotemporal domain. We train two networks to predict optical flow and ego-motion and depth, and use the predictions to attempt to remove the

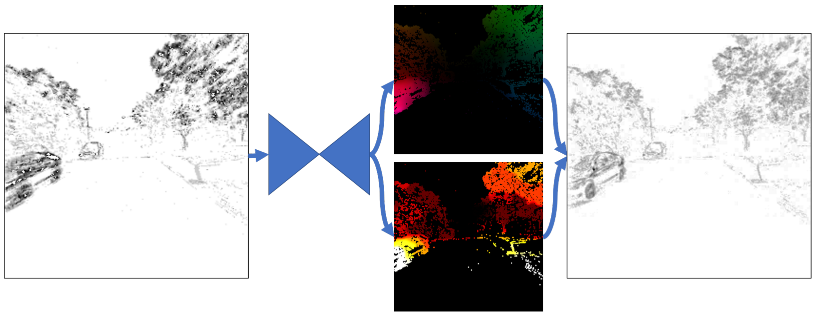


Figure 29: Our network learns to predict motion from motion blur by predicting optical flow (top) or egomotion and depth (bottom) from a set of input, blurry, events from an event camera (left), and minimizing the amount of motion blur after deblurring with the predicted motion to produce the deblurred image (right).

motion blur generated when the events are projected into the 2D image plane, as visualized in Fig. 29. Our unsupervised loss then measures the amount of motion blur in the corrected event image, which provides a training signal to the network. In addition, our deblurred event images are comparable to edge maps, and so we apply a stereo loss on the census transform of these images to allow our network to learn metric poses and depths. We evaluate both methods on the Multi Vehicle Stereo Event Camera dataset, Chapter 4, and compare against the equivalent grayscale based methods, as well as the method proposed in Chapter 6.

The contributions in this chapter can be summarized as:

- A novel discretized event volume for passing events into a neural network.
- A novel application of a motion blur based loss function that allows for unsupervised learning of motion information from events only.
- A novel stereo similarity loss applied on the census transform of a pair of deblurred event images.
- Quantitative evaluations on the Multi Vehicle Stereo Event Camera dataset, with qualitative and quantitative evaluations from a variety of night time and other challenging scenes.

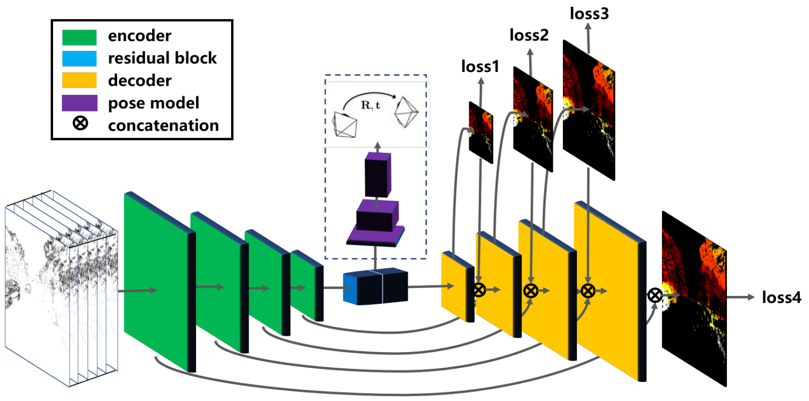


Figure 30: Network architecture for both the optical flow and egomotion and depth networks. In the optical flow network, only the encoder-decoder section is used, while in the egomotion and depth network, the encoder-decoder is used to predict depth, while the pose model predicts the egomotion. At training time, the loss is applied at each stage of the decoder, before being concatenated into the next stage of the network.

7.2 Related Work

Since the introduction of event cameras, such as Lichtsteiner et al. [84], there has been a strong interest in the development of algorithms that leverage the benefits provided by these cameras. In the work of optical flow, Benosman et al. [13] showed that normal flow can be estimated by fitting a plane to the events in x-y-t space. Bardow et al. [6] show that flow estimation can be written as a convex optimization problem that solves for the image intensity and flow jointly.

In the space of SFM and visual odometry, Kim et al. [74] demonstrate that a Kalman filter can reconstruct the pose of the camera and a local map. Rebecq et al. [119] similarly build a 3D map, which they localize from using the events. Zhu et al. [168] use an EM based feature tracking method to perform visual-inertial odometry, while Rebecq et al. [121] use motion compensation to deblur the event image, and run standard image based feature tracking to perform visual-inertial odometry.

For model-free methods, self-supervised and unsupervised learning have allowed deep

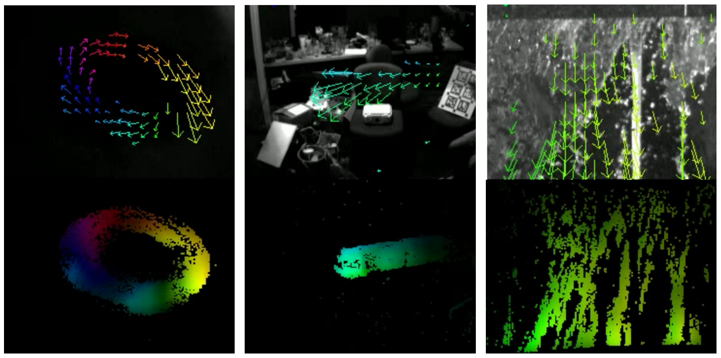


Figure 31: Our flow network is able to generalize to a variety of challenging scenes. Top images are a subset of flow vectors plotted on top of the grayscale image from the DAVIS camera, bottom images are the dense flow output of the network at pixels with events, colored by the direction of the flow. Left to right: Fidget spinner spinning at 13 rad/s in a very dark environment. Ball thrown quickly in front of the camera (the grayscale image does not pick up the ball at all). Water flowing outdoors.

networks to learn motion and the structure of a scene, using only well established geometric principles. Yu et al. [158] established that a network can learn optical flow from brightness constancy with a smoothness prior, while Meister et al. [93] extend this work by applying a bidirectional census loss to improve the quality of the flow. In a similar fashion, Zhou et al. [164] show that a network can learn a camera's egomotion and depth using camera reprojection and a photoconsistency loss. Zhan et al. [162] and Vijayanarasimhan et al. [145] add in a stereo constraint, allowing the network to learn absolute scale, while Wang et al. [146] apply this concept with a recurrent neural network.

In this work, we adapt a novel formulation of the motion blur loss from Mitrokhin et al. [96] for a neural network, by generating a single fully differentiable loss function that allows our networks to learn optical flow and structure from motion in an unsupervised manner.

7.3 Method

Our pipeline consists of a novel volumetric representation of the events, which we describe in Section 7.3.1, which is passed through a fully convolutional neural network to predict flow and/or egomotion and depth. We then use the predicted motion to try to deblur the events, and apply a loss that minimizes the amount of blur in the deblurred image, as described in Section 7.3.2. This loss can be directly applied to our optical flow network, Section 7.3.3. For the egomotion and depth network, we describe the conversion to optical flow in Section 7.3.4, as well as a novel stereo disparity loss in Section 7.3.4. Our architecture is summarized in Figure 30.

7.3.1 Input: The Discretized Event Volume

Selecting the appropriate input representation of a set of events for a neural network is still a challenging problem. Prior works such as Moeys et al. [97] and Maqueda et al. [90] generate an event image by summing the number of events at each pixel. However, this discards the rich temporal information in the events, and is susceptible to motion blur. Zhu et al. [173] and Ye et al. [156] propose image representations of the events, that summarize the number of events at each pixel, as well as the last timestamp and average timestamp at each pixel, respectively. Both works show that this is sufficient for a network to predict accurate optical flow. While this maintains some of the temporal information, a lot of information is still lost by summarizing the high resolution temporal information in the events.

We propose a novel input representation generated by discretizing the time domain. In order to improve the resolution along the temporal domain beyond the number of bins, we insert events into this volume using a linearly weighted accumulation similar to bilinear interpolation.

Given a set of N input events $\{(x_i, y_i, t_i, p_i)\}_{i\in[1,N]}$, and a set B bins to discretize the time dimension, we scale the timestamps to the range $[0, B-1]$, and generate the event volume as follows:

$$t_i^* =(B-1)(t_i - t_1)/(t_N - t_1) \tag{7.1}$$

$$V(x,y,t) =\sum_i p_i k_b(x-x_i)k_b(y-y_i)k_b(t-t_i^*) \tag{7.2}$$

$$k_b(a) = \max(0, 1-|a|) \tag{7.3}$$

where $k_b(a)$ is equivalent to the bilinear sampling kernel defined in Jaderberg et al. [69]. Note that the interpolation in the x and y dimensions is necessary when camera undistortion or rectification is performed, resulting in non integer pixel positions. In the case where no events overlap between pixels, this representation allows us to reconstruct the exact set of events. When multiple events overlap on a voxel, the summation does cause some information to be lost, but the resulting volume retains the distribution of the events across the spatiotemporal dimensions within the window.

In this work, we treat the time domain as channels in a traditional 2D image, and perform 2D convolution across the x, y spatial dimensions. We found negligible performance increases when using 3D convolutions, for a significant increase in processing time.

7.3.2 Supervision through Motion Compensation

As event cameras register changes in log intensity, the standard model of photoconsistency does not directly apply onto the events. Instead, several works have applied the

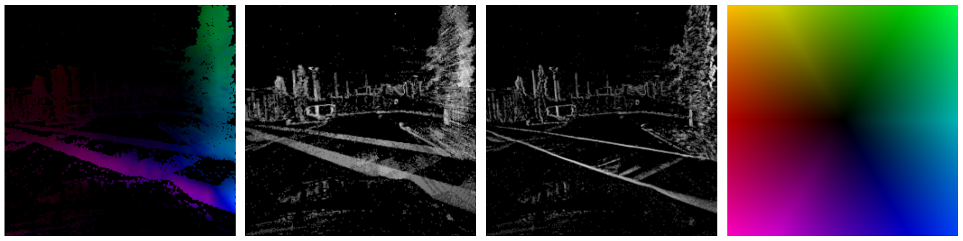


Figure 32: Our network learns to predict motion from motion blur by predicting optical flow or egomotion and depth (1) from a set of input, blurry, events (2), and minimizing the amount of motion blur after deblurring with the predicted motion to produce the deblurred image (3). The color of the flow indicates direction, as draw in the colorwheel (4).

concept of motion compensation, as described in Rebecq et al. [121], as a proxy for photoconsistency when estimating motion from a set of events. The goal of motion compensation is to use the motion model of each event to deblur the event image, as visualized in Figure 32.

For the most general case of per pixel optical flow, $u(x, y), v(x, y)$, we can propagate the events, $\{(x_i, y_i, t_i, p_i)\}_{i=1,\dots,N}$, to a single time t':

$$\begin{pmatrix} x'_i \\ y'_i \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \end{pmatrix} + (t' - t_i) \begin{pmatrix} u(x_i, y_i) \\ v(x_i, y_i) \end{pmatrix} \tag{7.4}$$

If the input flow is correct, this reverses the motion in the events, and removes the motion blur, while for an incorrect flow, this will likely induce further motion blur.

We use a measure of the quality of this deblurring effect as the main supervision for our network. Gallego et al. [45] proposed using the image variance on an image generated by the propagated events. However, we found that the network would easily overfit to this loss, by predicting flow values that push all events within each region of the image to a line.

Instead, we adopt the loss function described by Mitrokhin et al. [96], who use a loss which maximizes the variance of the timestamps at each pixel. This, as shown below, is equivalent to minimizing the average timestamp squared, at each pixel.

$$\max_{u,v} \sum_{x,y} \mathrm{var}(\{t_j\}), j \in \{j | x = x_j + t_j u, y = y_j + t_j v\} \tag{7.5}$$

$$\equiv \max_{u,v} \sum_{x,y} \left(\sum_j t_j^2 \right) - \left(\frac{1}{n} \sum_j t_j \right)^2 \tag{7.6}$$

As the number of events remain fixed, the first term, corresponding to the sum of all timestamps, is a constant, and does not affect the optimization.

$$\equiv \max_{u,v} - \sum_{x,y} \left(\frac{1}{n} \sum_j t_j \right)^2 \tag{7.7}$$

$$\equiv \min_{u,v} \sum_{x,y} \left(\frac{1}{n} \sum_j t_j \right)^2 \tag{7.8}$$

However, the previously proposed loss function is non-differentiable, as the timestamps were rounded to generate an image. To resolve this, we replace the rounding with bilinear interpolation. We apply the loss by first separating the events by polarity and generating an image of the average timestamp at each pixel for each polarity, T_+, T_-:

$$\begin{aligned} T_{p'}(x, y | t') =& \frac{\sum_i \mathbb{1}(p_i = p') k_b(x - x_i') k_b(y - y_i') t_i}{\sum_i \mathbb{1}(p_i = p') k_b(x - x_i') k_b(y - y_i') + \epsilon} \\ p' \in& \{+, -\}, \epsilon \approx 0 \end{aligned} \tag{7.9}$$

The loss is, then, the sum of the two images squared.

$$\mathcal{L}_{\text{time}}(t') = \sum_x \sum_y T_+(x,y|t')^2 + T_-(x,y|t')^2 \tag{7.10}$$

However, using a single t' for this loss poses a scaling problem. In (7.4), the output flows, u, v, are scaled by $(t' - t_i)$. During backpropagation, this will weight the gradient over events with timestamps further from t' higher, while events with timestamps very close to t' are essentially ignored. To mitigate this scaling, we compute the loss both backwards and forwards, with $t' = t_1$ and $t' = t_N$:

$$\mathcal{L}_{\text{time}} = \mathcal{L}_{\text{time}}(t_1) + \mathcal{L}_{\text{time}}(t_N) \tag{7.11}$$

Note that changing the target time, t', does not change the timestamps used in (7.9).

This loss function is similar to that of Benosman et al. [13], who model the events with a function Σ_{e_i}, such that $\Sigma_{e_i}(\mathbf{x}_i) = t_i$. In their work, they assume that the function is locally linear, and solve the minimization problem by fitting a plane to a small spatiotemporal window of events. We can see that the gradient of the average timestamp image, $(dt/dx, dt/dy)$, corresponds to the inverse of the flow, if we assume that all events at each pixel have the same flow.

7.3.3 Optical Flow Prediction Network

Using the input representation and loss described in Section 7.3.1 and 7.3.2, we train a neural network to predict optical flow. We use an encoder-decoder style network, as in [173]. The network outputs flow values in units of pixels/bin, which we apply to (7.4), and eventually compute (7.13).

Our flow network uses the temporal loss in (7.11), combined with a local smoothness

	outdoor day1		indoor flying1		indoor flying2		indoor flying3	
dt=1 frame	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier
Ours	**0.32**	**0.0**	0.58	**0.0**	1.02	4.0	0.87	3.0
EV-FlowNet	0.49	0.2	1.03	2.2	1.72	15.1	1.53	11.9
UnFlow	0.97	1.6	**0.50**	0.1	**0.70**	**1.0**	**0.55**	**0.0**

	outdoor day1		indoor flying1		indoor flying2		indoor flying3	
dt=4 frames	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier	AEE	% Outlier
Ours	1.30	9.7	**2.18**	**24.2**	**3.85**	46.8	3.18	47.8
EV-FlowNet	**1.23**	**7.3**	2.25	24.7	4.05	**45.3**	3.45	39.7
UnFlow	2.95	40.0	3.81	56.1	6.22	79.5	**1.96**	**18.2**

Table 6: Quantitative evaluation of our optical flow network compared to EV-FlowNet and UnFlow. For each sequence, Average Endpoint Error (AEE) is computed in pixels, % Outlier is computed as the percent of points with AEE > 3 pix. dt=1 is computed with a time window between two successive grayscale frames, dt=4 is between four grayscale frames.

regularization:

$$\mathcal{L}_{\text{smooth}} = \sum_{\vec{x}} \sum_{\vec{y} \in \mathcal{N}(\vec{x})} \rho(u(\vec{x}) - u(\vec{y})) + \rho(v(\vec{x}) - v(\vec{y})) \tag{7.12}$$

where $\rho(x) = \sqrt{x^2 + \epsilon^2}$ is the Charbonnier loss function [29], and $\mathcal{N}(x, y)$ is the 4-connected neighborhood around (x, y).

The total loss for the flow network is:

$$\mathcal{L}_{\text{flow}} = \mathcal{L}_{\text{time}} + \lambda_1 \mathcal{L}_{\text{smooth}} \tag{7.13}$$

7.3.4 Egomotion and Depth Prediction Network

We train a second network to predict the egomotion of the camera and the structure of the scene, in a similar manner to [162, 145]. Given a pair of time synchronized discretized event volumes from a stereo pair, we pass each volume into our network separately, but use both at training time to apply a stereo disparity loss, allowing our network to learn metric scale. We apply a temporal timestamp loss defined in Section 7.3.2, and a robust similarity loss between the census transforms [159, 137] of the deblurred event images.

The network predicts Euler angles, (ψ, β, ϕ), a translation, T, and the disparity of each pixel, d_i. The disparities are generated using the same encoder-decoder architecture as in the flow network, except that the final activation function is a sigmoid, scaled by the image width. The pose shares the encoder network with the disparity, and is generated by strided convolutions which reduce the spatial dimension from 16×16 to 1×1 with 6 channels.

Temporal Reprojection Loss

Given the network output, the intrinsics of the camera, K, and the baseline between the two cameras, b, the optical flow, (u_i, v_i) of each event at pixel location (x_i, y_i) is:

$$\begin{pmatrix} x_i^* \\ y_i^* \end{pmatrix} = K\pi\left(R\frac{fb}{d_i}K^{-1}\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} + T \right) \tag{7.14}$$

$$\begin{pmatrix} u_i \\ v_i \end{pmatrix} = \frac{1}{B-1}\left(\begin{pmatrix} x_i^* \\ y_i^* \end{pmatrix} - \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) \tag{7.15}$$

where f is the focal length of the camera, R is the rotation matrix corresponding to (ψ, β, ϕ) and π is the projection function: $\pi\left(\begin{pmatrix} X & Y & Z \end{pmatrix}^T\right) = \begin{pmatrix} \frac{X}{Z} & \frac{Y}{Z} \end{pmatrix}^T$. Note that, as the network only sees the discretized volume at the input, it does not know the size of the time window. As a result, the optical flow we compute is in terms of pixels/bin, where B is the number of bins used to generate the input volume. The optical flow is then inserted into (7.4) for the loss.

Stereo Disparity Loss

From the optical flow, we can deblur the events from the left and right camera using (7.4), and generate a pair of event images, corresponding to the number of events at each pixel after deblurring. Given correct flow, these images represent the edge maps of the corresponding grayscale image, over which we can apply a photometric loss. However, the number of events between the two cameras may also differ, and so we apply a similarity loss on the census transforms [159] of the images. For a given window width, W, we encode each pixel with a W^2 length vector, where each element is the sign of the difference between the pixel and each neighbor inside the window. For the left event volume, the right census transform is warped to the left camera using the left predicted disparities, and we apply a Charbonnier loss [29] on the difference between the two images, and vice versa for the right. In addition, we apply a left-right consistency loss between the two predicted disparities, as defined by [52]. Finally, we apply a local smoothness regularizer to the disparity, as in (7.12). The total loss for the SFM model is:

$$\mathcal{L}_{SFM} = \mathcal{L}_{temporal} + \lambda_2 \mathcal{L}_{stereo} + \lambda_3 \mathcal{L}_{consistency} + \lambda_4 \mathcal{L}_{smoothness} \tag{7.16}$$

7.4 Experiments

7.4.1 Implementation Details

We train two networks on the full outdoor_day2 sequence from MVSEC [170], which consists of 11 mins of stereo event data driving through public roads. At training, each input consists of $N = 30000$ events, which are converted into discretized event

	Threshold distance	10m	20m	30m
Sequence	Method	Average depth Error (m)		
outdoor_day1	Ours	**2.72**	**3.84**	**4.40**
	Monodepth	3.44	7.02	10.03
outdoor_night1	Ours	**3.13**	**4.02**	**4.89**
	Monodepth	3.49	6.33	9.31
outdoor_night2	Ours	**2.19**	**3.15**	**3.92**
	Monodepth	5.15	7.8	10.03
outdoor_night3	Ours	**2.86**	**4.46**	**5.05**
	Monodepth	4.67	8.96	13.36

Table 7: Quantitative evaluation of our depth network compared to Monodepth [52]. The average depth error is provided for all points in the ground truth up to 10m, 20m and 30m, with at least one event.

volumes with resolution 256x256 (centrally cropped) and $B = 9$ bins. The weights for each loss are: $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\} = \{1.0, 1.0, 0.1, 0.2\}$.

7.4.2 Optical Flow Evaluation

We tested our optical flow network on the indoor_flying and outdoor_day sequences from MVSEC, with the ground truth provided by [173]. Flow predictions were generated at each grayscale frame timestamp, and scaled to be the displacement for the duration of 1 grayscale frame (dt=1) and 4 grayscale frames (dt=4), separately. For the outdoor_day sequence, each set of input events was fixed at 30000, while for indoor_flying, 15000 events were used due to the larger motion in the scene. For comparison against ground truth, we convert our output, (u, v), from units of pixels/bin into units of pixel displacement with the following: $(\hat{u}, \hat{v}) = (u, v) \times (B - 1) \times dt/(t_N - t_0)$.

We present the average endpoint error (AEE), and the percentage of points with AEE greater than 3 pixels, over pixels with valid ground truth flow and at least one event. These results can be found in Tab. 6, where we compare our results against

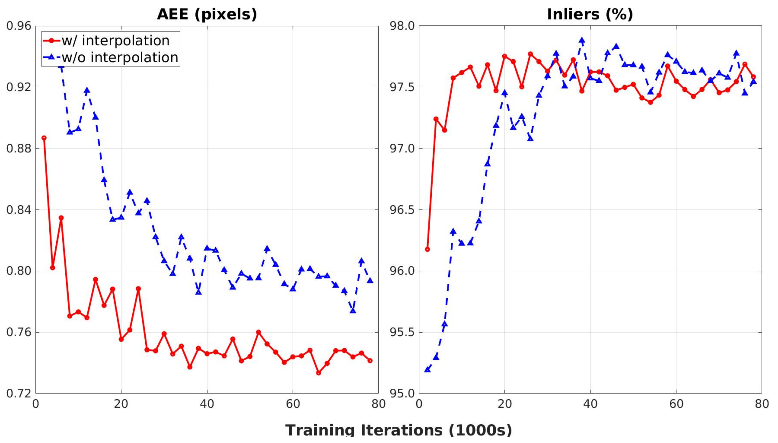


Figure 33: Ablation study on the effects of interpolation on the event volume. Flow prediction errors are shown against a held out validation set on two models with fixed random seed, with and without interpolation.

EV-FlowNet [173] and the image method UnFlow [93]. We do not provide results from ECN [156]. As their model assumes a rigid scene, and predicts egomotion and depth, they train on 80% of the indoor_flying sequences, and test on the other 20%. These results thus do not pose a fair comparison to our method, which is only trained on outdoor_day2. We do note that their outdoor_day1 errors are slightly lower than ours, at 0.30 vs 0.32. However, we believe that our method is more general, as it does not rely on a rigid scene assumption.

7.4.3 Egomotion Evaluation

We evaluate our ego-motion estimation network on the outdoor_day1 sequence from MVSEC. As there is currently no public code to the extent of our knowledge for unsupervised deep SFM methods with a stereo loss, we compare our ego-motion results against SFMLearner [164], and ECN [156], which learn egomotion and depth from monocular images and events. We train the SFMLearner models on the VI-Sensor images from the outdoor_day2 sequence, once again cropping out the hood of the car. These images are of a higher resolution than the DAVIS images, but are from

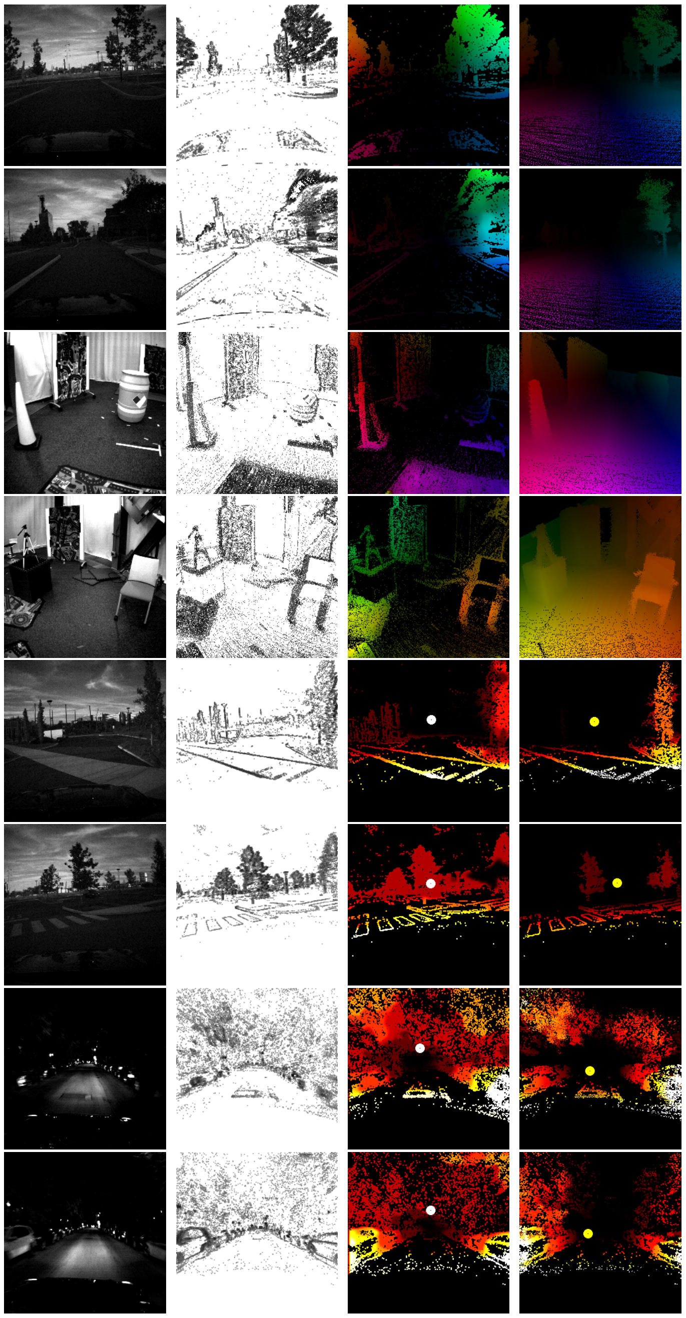


Figure 34: Qualitative results from the flow and egomotion and depth networks on the indoor flying, outdoor day and outdoor night sequences. From left to right: Grayscale image, event image, depth prediction with heading direction, ground truth with heading direction. Top four are flow results, bottom four are depth results. For depth, closer is brighter. Heading direction is drawn as a circle. In the outdoor night results, the heading direction is biased due to events generated by flashing lights.

Sequence	Method	Abs Rel	RMSE log	SILog	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
outdoor_day1	Ours	0.36	0.41	0.16	0.46	0.73	0.88
	ECN	**0.33**	**0.33**	**0.14**	**0.97**	**0.98**	**0.99**
outdoor_night	Ours	**0.37**	**0.42**	**0.15**	0.45	0.71	0.86
	ECN	0.39	**0.42**	0.18	**0.95**	**0.98**	**0.99**

Table 8: Quantitative evaluation of standard depth metrics from our depth network against ECN [156]. Left to right, the metrics are: absolute relative distance, RMSE log, scale invariant log, and the percentage of points with predicted depths beyond 1.25, 1.25^2 and 1.25^3 times larger or smaller than the ground truth.

	ARPE (deg)	ARRE (rad)
Ours	7.74	0.00867
SFM Learner [164]	16.27	0.00939
ECN [156]	**3.98**	**0.000267**

Table 9: Quantitative evaluation of our egomotion network compared to SFM Learner. ARPE: Average Relative Pose Error. ARRE: Average Relative Rotation Error.

the same scene, and so should generalize as well as training on the DAVIS images. The model is trained from scratch for 100k iterations. As the translation predicted by SFMLearner is only up to a scale, we present errors in terms of angular error. The relative pose errors (RPE) and relative rotation errors (RRE) are computed as: $\text{RPE} = \arccos\left(\frac{t_{\text{pred}} \cdot t_{\text{gt}}}{\|t_{\text{pred}}\|_2 \|t_{\text{gt}}\|_2}\right)$, $\text{RRE} = \|\text{logm}(R_{\text{pred}}^T R_{\text{gt}})\|_2$, where R_{pred} is the rotation matrix corresponding to the Euler angles from the output, and logm is the matrix logarithm.

7.4.4 Depth Network Evaluation

We compare our depth results against Monodepth [52], which learns monocular disparities from a stereo pair at training time. As the DAVIS grayscale images are not time synchronized, we train on the cropped VI-Sensor images. The model is trained for 50 epochs, and we provide depth errors with thresholds up to 10m, 20m and 30m in the ground truth and with at least one event. In Tab. 8, we provide the scale invariant depth metrics reported by ECN [156].

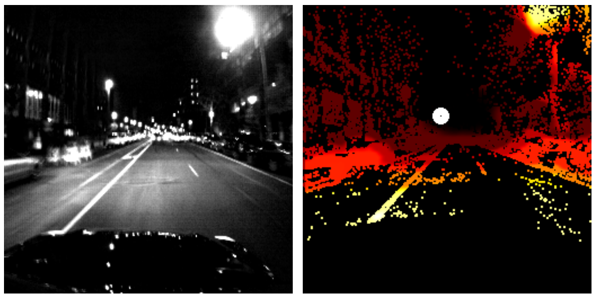


Figure 35: Failure case of our depth network. The flashing street light is detected as very close due to spurious events.

7.4.5 Event Volume Ablation

To test the effects of the proposed interpolation when generating the discretized event volume, we provide results in Fig. 33 of flow validation error during training between a model with and without interpolation. These results show that, while both models are able to converge to accurate flow estimates and similar % outliers, the interpolated volume achieves lower AEE.

7.5 Results

7.5.1 Optical Flow

From the quantitative results in Tab. 6, we can see that our method outperforms EV-FlowNet in almost all experiments, and nears the performance of UnFlow on the short 1 frame sequences. Qualitative results can be found in Fig. 34.

In general, we have found that our network generalizes to a number of very different and challenging scenes, including those with very fast motions and dark environments. A few examples of this can be found in Fig. 31. We believe this is because the events do not have the fine grained intensity information at each pixel of traditional images, and so there is less redundant data for the network to overfit.

7.5.2 Egomotion

Our model trained on outdoor_day2 was able to generalize well to outdoor_day1, despite the environment changing significantly from an outdoor residential environment to a closed office park area. In Tab. 7, we show that our relative pose and rotation errors are significantly better than that of SFM-Learner, but worse than ECN. However, ECN only predicts 5dof pose, up to a scale factor, while our network must learn the full 6dof pose with scale. We believe that additional training data may bridge this gap.

As the network was only trained on driving sequences, we were unable to achieve good egomotion generalization to the outdoor_night sequences. We found that this was due to the fluorescent lamps found at night, which generated many spurious events due to their flashing that were not related to motion in the scene. As our egomotion network takes in global information in the scene, it tended to perceive these flashing lights as events generated by camera motion, and as a result generated an erroneous egomotion estimate. Future work to filter these kinds of anomalies out will be necessary. For example, if the rate of the flashing is known a-priori, the lights can be simply filtered by detecting events generated at the desired frequency.

7.5.3 Depth

Our depth model was able to produce good results for all of the driving sequences, although it is unable to generalize to the flying sequences. This is likely because the network must memorize some concept of metric scale, which cannot generalize to completely different scenes. We outperform Monodepth in all of the sequences, which is likely because the events do not have intensity information, so the network is forced to learn geometric properties of objects. In addition, the network generalizes well even in the face of significant noise at night, although flashing lights cause the network to predict very close depths, such as in Fig. 35.

For the scale invariant metrics in Tab. 8, our method is comparable to ECN [156] in most errors, despite having to predict absolute scale, whereas the depths in ECN are corrected for scale. However, our δ percentages are lower than expected. We believe that additional training data can alleviate this issue in the future.

7.6 Conclusions

In this chapter, we demonstrate a novel input representation for event cameras, which, when combined with our motion compensation based loss function, allows a deep neural network to learn to predict optical flow and ego-motion and depth from the event stream only. This allows for extremely simple data collection, simply by recording data from the event stream as the camera is moved.

Chapter 8

EventGAN: Leveraging Large Scale Image Datasets for Event Cameras

8.1 Introduction

In the previous two chapters, we have applied self-supervised learning methods for events in order to avoid the problem of having to collect labeled training data. While this was successful for geometric problems, there is a large class of problems for which no self-supervised approach currently exists. In particular many recognition tasks, which are perhaps the most successful computer vision problems solved to date, require a large corpus of labeled training data. In this work, we focus on an alternative to costly data labeling, by leveraging the large set of existing labeled image datasets for events via image to event simulation.

The highest fidelity event camera simulators today [123, 103, 83] all operate with a similar framework, by simulating optical flow in the image either through 3D camera motion, or a parametrized warping (e.g. affine) of the image, in order to precisely track the generation of events as each point in the image moves to a new pixel. However, these scenarios either require simulation of the full 3D scene, or severely constrain the motion in the image. In addition, modeling event noise, both in terms of erroneous events and noise in the event measurements, is a challenging open problem.

In this work, we present EventGAN, a novel method for image to event simulation, where we apply a convolutional neural network as the function between images and events. By learning this function with data, our method does not require any explicit knowledge of the scene or the relationship between images and events, but is instead able to regress a realistic set of events given only images as input. In addition, our network is able to learn the noise distribution over the events, which are currently not modeled by the competing methods. Finally, our proposed method has a fast, constant time simulation which is easily parallelizable on GPUs and integrable into any modern neural network architecture, as opposed to the prior work which requires 3D simulations of the scene.

Our network is trained on a set of image and event pairs, which are directly output by event cameras such as the DAVIS [18]. At training time, we apply an adversarial loss to align the generated events with the real events. In addition, we pre-train a pair of CNNs to perform optical flow estimation and image reconstruction from real events, and constrain our generator to produce events which allow these pre-trained networks to generate accurate outputs. In other words, we constrain the generated events to retain the motion and appearance information present in the real data.

Using this event simulation network, we train a set of downstream networks to perform object detection on cars and 2D human pose estimation, given images and labels from large scale image datasets such as KITTI [48], MPII [4] and Human3.6M [67]. We then evaluate performance on these downstream tasks on real event datasets, MVSEC [171] for car detection, and DHP19 [21] for human pose, demonstrating the generalization ability of these networks despite having mostly seen simulated data at training time. All data and code will be released at a later date.

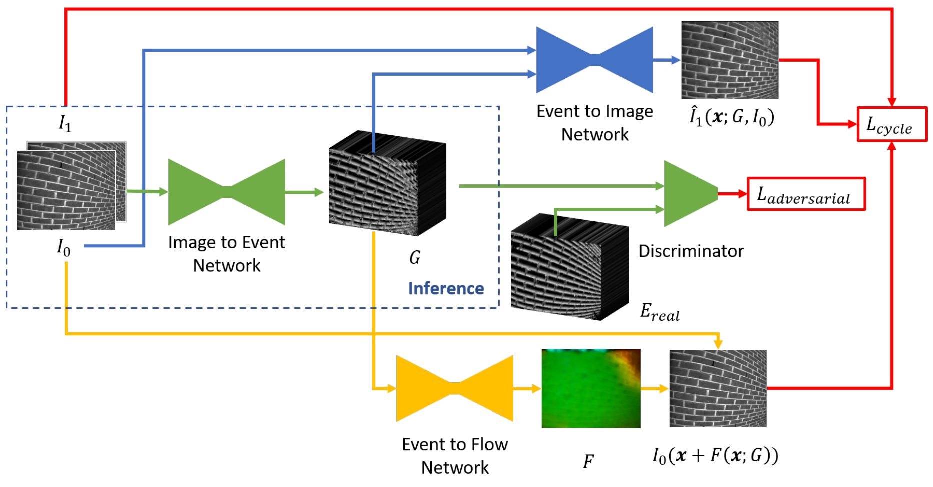


Figure 36: Overview of the EventGAN pipeline. A pair of grayscale images are passed into the generator, which predicts a corresponding event volume. This output is constrained by an adversarial loss, as well as a pair of cycle consistency losses which constrain the generated volume to encode image and flow information.

Our main contributions can be summarized as:

- A novel pipeline for supervised training of deep neural networks for events, by simulating events from existing large scale image datasets and training on the simulated events and image labels.
- A novel network, EventGAN, for event simulation from a pair of images, trained using an adversarial loss and cycle consistency losses which constrain the generator network to generate events from which pre-trained networks are able to extract accurate optical flow and image reconstructions.
- A test dataset for car detection, with manually labeled bounding boxes for cars from the MVSEC [171] dataset.
- Experiments demonstrating the generalizability of the networks trained on simulated data to real event data, by training object detection and human pose networks on simulated data, and evaluating on real data.

8.2 Related Work

8.2.1 Event Simulation

Prior works on event simulation have focused on differencing log intensity frames, in order to simulate the condition required to trigger an event:

$$\|\log(I_{t+1}(\mathbf{x})) - \log(I_t(\mathbf{x}))\| \geq \theta \tag{8.1}$$

Earlier works by Bi et al. [15] and Kaiser et al. [71] simulating events by directly applying this equation to the log intensity difference between each pair of successive images. These methods were limited by the temporal resolution of these images, and as such could only handle relatively slow moving scenes. To improve fidelity, Rebecq et al. [123], Mueggler et al. [103] and Li et al. [83] perform full 3D simulations of a scene. This allows them to simulate images at arbitrary temporal resolution, while also having access to the optical flow within the scene, allowing for accurate event trajectories. However, these methods are limited to fully simulated scenes, or images where the motion is known (or where a simplified motion model such as an affine transform is applied). Performing 3D simulations is also a relatively expensive procedure, requiring complex rendering engines. In addition, these methods do not properly model the noise properties of the sensor. Rebecq et al. [123] apply Gaussian noise to the trigger threshold, θ, as an approximation, but no true model of the event noise distribution exists to our knowledge.

Our work, in contrast, runs in constant time using a CNN which is easily parallelizable and optimized for modern GPUs. The network learns both the motion information in the scene, as well as the noise distribution of the events.

8.2.2 Sim2Real/Domain Adapation

Learning from simulations and other modalities has been a rapidly growing topic, with deep learning approaches for many robotics problems in particular requiring much more training data than is practical to collect on a physical platform. However, this remains a challenging open problem, as conventional simulators often cannot perfectly model the data distribution in the real world, resulting in many methods attempting to bridge this gap [113, 70]. One popular approach to this problem in the image space is the use of Generative Adversarial Networks (GANs) [53], which consists of a generator trained to model the data distribution of the training set, while a discriminator is trained to differentiate between outputs from the fake and real data. With particular relevance to this work, conditional GANs [95, 68] are able to model relationships between data distributions, while CycleGANs apply additional cycle consistency losses [175].

A successful application of cross modality transfer is in the field of image to lidar transform. A number of recent works [157, 147, 152] have approached the problem of simulating lidar measurements from images, which allow networks to better reason about 3D scenes more efficiently.

With a similar motivation to our work, Iacono et al. [66] and Zanardi et al. [160] address the issue of transferring learning from images to events by running a network trained on images on the grayscale images produced by some event cameras such as the DAVIS [18], and using these outputs as ground truth to train a similar network for events. However, these methods treat the frame based outputs as ground truth, and so will learn biases and mistakes made by the frame based network (e.g. the best mAP of the grayscale network in Zanardi et al. [160] is 0.59, resulting in a mAP for the event based network of 0.26).

As an alternative approach, our work follows the philosophy of using GANs for image to event simulation. We then use the simulated events to train directly on the ground truth labels for the corresponding images, which should be at least as accurate if not better than outputs from a frame based network trained on these labels.

8.3 Method

The generative portion of our pipeline consists of a U-Net [129] encoder-decoder network, as used in Zhu et al. [174] and Rebecq et al. [124]. The generator takes as input a pair of grayscale images, concatenated along the channel dimension, and outputs a volumetric representation of the events, described in Section 8.3.1. To constrain this output, we apply an adversarial loss, described in Section 8.3.2, as well as a pair of cycle consistency losses, described in Section 8.3.3. The full pipeline for our method can be found in Figure 36.

8.3.1 Event Representation

The most compact way to represent a set of events is as a set of 4-tuples, consisting of the x, y position, timestamp, t, and polarity, p. However, regressing points in general is a difficult task, and faces challenges such as varying numbers of events and permutation invariance.

In this work, we bypass this issue by instead regressing an intermediate representation of the events as proposed by Zhu et al. [174]. In this representation, the events are scattered into a fixed size 3D spatiotemporal volume, where each event, (x, y, t, p) is inserted into the volume, which has $B = 9$ temporal channels, with a linear kernel:

$$t_i^* =(B-1)(t_i - t_1)/(t_N - t_1) \tag{8.2}$$

$$V(x,y,t) = \sum_i \max(0, 1 - |t - t_i^*|) \tag{8.3}$$

This retains the distribution of the events in x-y-t space, and has shown success in a number of tasks [174, 28, 124].

However, we deviate from the prior work in that we generate separate volumes for each polarity, and concatenate them along the time dimension. This results in a volume which is strictly non-negative, allowing for a ReLU as the final activation of the network, such that the sparsity in the volume is easily preserved.

In addition, we normalize this volume similar to Rebecq et al. [124], with an additional clipping step, as follows:

$$\hat{V}(x,y,t) = \frac{\min(V(x,y,t), \eta_{98})}{\eta_{98}} \tag{8.4}$$

where η_{98} is the 98th percentile value in the set of non-zero values of V. This equates to a clipping operation, followed by a normalization such that the volume lies in $[0,1]$. The clipping is designed to reduce the effect of hot pixels, which have an erroneously low contrast thresholds and thus generate a disproportionately many events, skewing the range.

8.3.2 Adversarial Loss

Perhaps the most direct way to supervise this network is to apply a direct numerical error, such as a L1 or L2 loss, between the predicted and real events. However, given a pair of images, the number of plausible event distributions between the images is

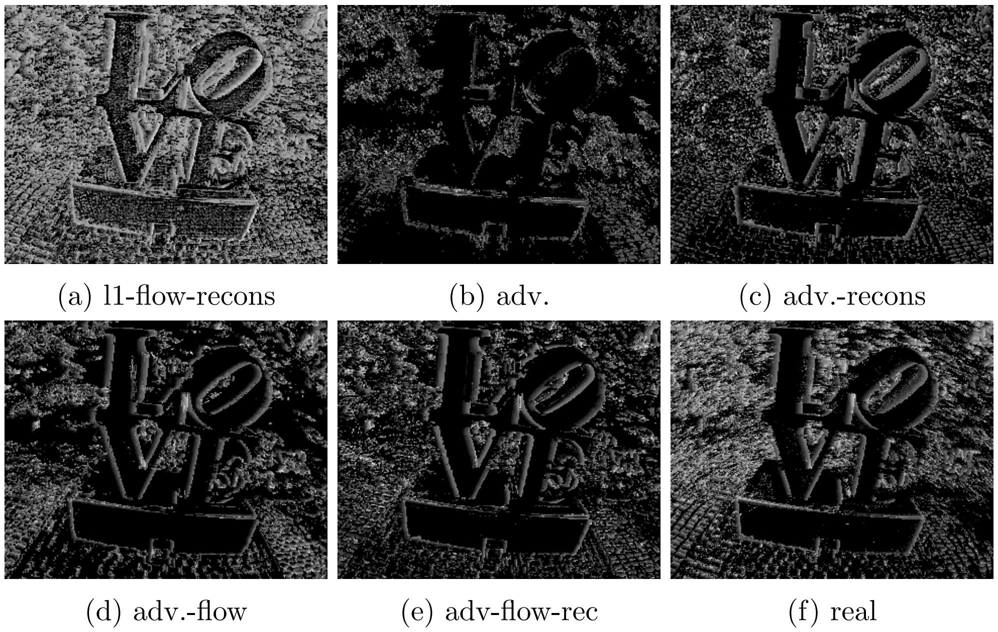

(a) l1-flow-recons (b) adv. (c) adv.-recons (d) adv.-flow (e) adv-flow-rec (f) real

Figure 37: Outputs from models trained with subsets of our proposed loss, all with the same hyperparameters. Events are visualized as average timestamp images, i.e. the average timestamp at each pixel. Any voxel with non zero value will generate a color in the average timestamp image, allowing us to see the sparsity of the volume. **(a)**: L1 reconstruction loss in place of the adversarial loss, causing artifacts in the events, and no sparsity achieved, as observed in the interior of the 'LOVE' symbol in the time image. **(b)**: Adversarial loss only. Model struggles to converge, and requires significant hyperparameter tuning in order to achieve good results. **(c)**: Adversarial loss and reconstruction loss. Model is now stable, but the events do not have motion information. The image should have a gradient in the motion direction. **(d)**: Adversarial loss and flow loss. Motion direction can now be seen in the time image, but events are not generated in many areas. **(e)**: Adversarial loss, flow and reconstruction losses. Motion trails can now be clearly seen in the time image (see letters). **(f)**: Real events. Note that our method typically underestimates the amount of motion in the scene.

extremely large (two images can not constrain the exact motion in between them). Such a direct loss would likely cause the network to overfit to the trajectories observed in the training set and fail to generalize.

Instead, we apply an adversarial loss [53]. This loss simply constrains the generated events to follow the same distribution as the real ones, and avoids directly constraining the network to memorizing the trajectories seen at training time. For each event-image pair, (x, y), we regress a generated event volume using our network, G, and then pass the generated events and real events through a discriminator network, D, which predicts the probability that its input is from real data. Our discriminator is a 4 layer PatchGAN classifier [68]. We alternatingly train the generator and discriminator, with the discriminator trained 2 steps for every 1 of the generator, using the hinge adversarial loss [85, 141]:

$$\begin{aligned}\mathcal{L}_D =& -\mathbb{E}_{(x,y)\sim p_{\text{data}}}[\min(0, -1 + D(x, y))] \\ & -\mathbb{E}_{y\sim p_{\text{data}}}[\min(0, -1 - D(G(y), y))] \qquad (8.5)\\ \mathcal{L}_G =& -\mathbb{E}_{y\sim p_{\text{data}}} D(G(y), y) \qquad (8.6)\end{aligned}$$

8.3.3 Cycle Consistency Losses

However, GANs are typically difficult to train, especially with a high dimensional output space such as an event volume. In addition, there are no guarantees on the simulated events retaining the salient information in the images, such as accurate motion and intensity information.

To this end, we apply an additional pair of losses which constrain the generated events to encode this motion and intensity information. In particular, we pre-train a pair of networks for optical flow estimation and image reconstruction from real events, using the pipeline in EV-FlowNet [173].

The flow network takes as input the event volume, and outputs a per pixel optical flow. Supervision is applied by warping the previous image to the time of the next image using the predicted flow, and applying an L1 loss between the warped and original image, as well as a local smoothness constraint.

The image reconstruction network takes as input the previous image and the event volume, and outputs the predicted next image, and is directly supervised by a L1 loss between the reconstructed and original image. The previous image is provided as input as we found that the image reconstruction network tended to overfit to the training set without it. Prior work by Rebecq et al. [124] has circumvented this by training in a recurrent fashion, but doing so would require multiple passes through the recurrent network, which is undesirably expensive when the goal is to train the generator network. In addition, we summarize the event volume by summing along the time dimension. This is to maintain the invariance to permutation across time of the events. For example, two events occurring at the start of the window vs. two events at the end of the window should generate the same output image. The input, then, to the reconstruction network, is a 2-channel image consisting of the previous image and the summed event volume.

In summary, the cycle consistency losses are:

$$\begin{aligned}
\mathcal{L}_F =& \sum_{\mathbf{x}} \|I_0(\mathbf{x} - F(\mathbf{x}; G)) - I_1(\mathbf{x})\|_1 \\
&+ \lambda_1 \left(\left\| \frac{dF}{dx}(\mathbf{x}; G) \right\|_1 + \left\| \frac{dF}{dy}(\mathbf{x}; G) \right\|_1 \right) && (8.7) \\
\mathcal{L}_R =& \sum_{\mathbf{x}} \|\hat{I}_1(\mathbf{x}; G, I_0) - I_1(\mathbf{x})\|_1 && (8.8) \\
\mathcal{L}_{\text{cycle}} =& \mathcal{L}_F + \mathcal{L}_G && (8.9)
\end{aligned}$$

When training the generator network, we pass the output from the generator as input to each of the pre-trained networks, and apply the same losses used to train each. However, in this case, we freeze the weights of each pre-trained network, such that the generator must tune its output to generate the best input for each pre-trained network. Both cycle consistency networks share the same architecture as the generator network, with the losses applied each time the generator is updated in the adversarial framework. The final losses at each step are:

$$\text{Generator step:} \qquad \mathcal{L}_{GS} = \mathcal{L}_G + \mathcal{L}_{\text{cycle}} \tag{8.10}$$

$$\text{Discriminator step:} \qquad \mathcal{L}_{DS} = \mathcal{L}_D \tag{8.11}$$

These losses provide useful gradients early in training, when the adversarial loss is typically unstable, and embed motion and appearance information in the predicted event volumes. Figure 37 shows the effect of each loss on the output of the generator.

In summary, the adversarial loss enforces sparsity in the event volume and similarity between the fake and real event distributions. The flow loss enforces motion information to be present within the volume, while the reconstruction loss enforces regularity in the number of events generated by the same point. This is particularly evident when one visualizes the image of the average timestamp at each pixel, where extremely low (but non-zero) values may be hidden in the count image, and where motion trails are clearly visible.

8.4 Experiments

We train our network on events and images from the indoor_flying and outdoor_day sequences in the MVSEC dataset [171], as well as a newly collected dataset consisting of recordings from a DAVIS-346b camera [18], consisting of short (<60s) sequences with a number of different scenes and motions, in order to capture a large range of event distributions. As the objective of this work is to produce an event simulator which operates well on existing image datasets, we did not train on scenes which are

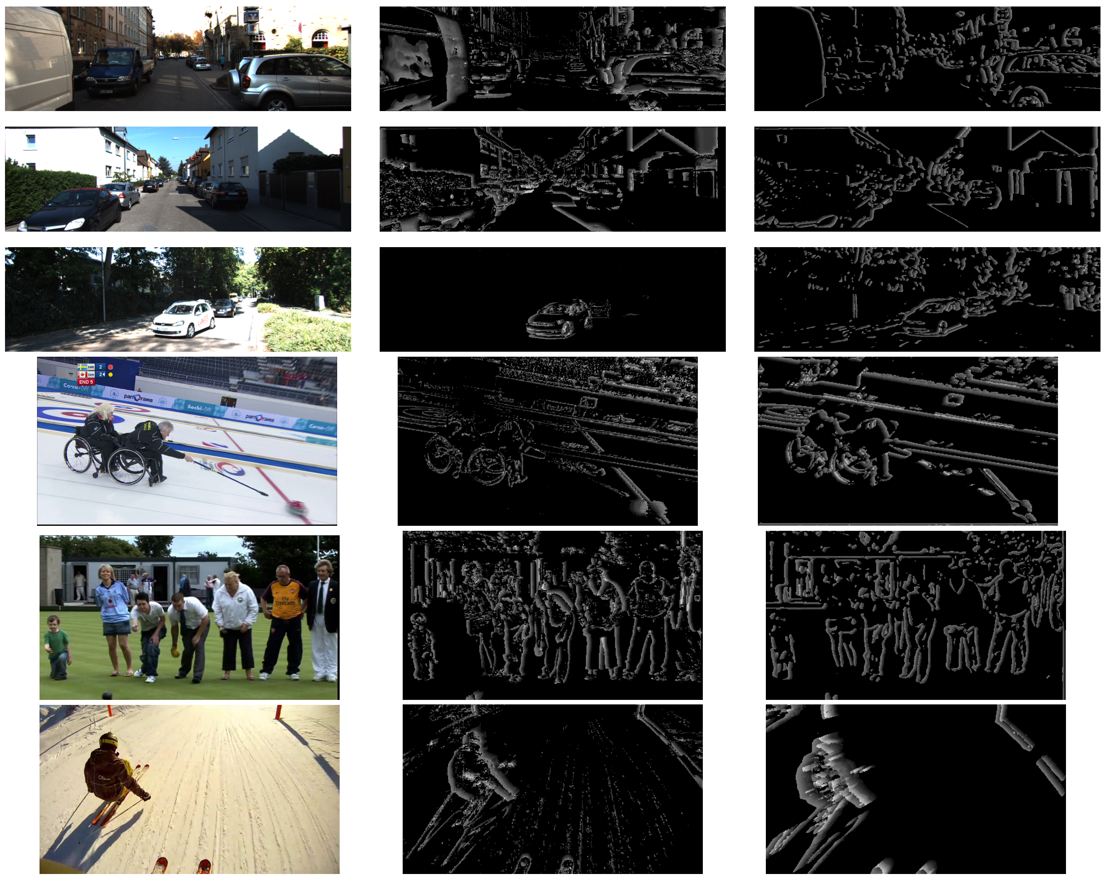


Figure 38: Sample outputs generated by EventGAN, compared to ESIM [124], visualized as images of the average timestamp at each pixel. Top images are from KITTI [48], bottom are from MPII [4]. Compared to ESIM, our method is able to more accurately capture the motion in the scene, and capture fine grain information.

challenging for images (e.g. night time driving). In total, the training set consists around 30 mins of data. During training, we perform weighted sampling from this dataset, with a 80%/20% split between the new data and MVSEC. Each input to the network consists of a pair of images, randomly picked between 1 and 6 frames apart, and the events between them.

Quantitative evaluations of generative models is difficult, as measuring how well the predicted events fit the true event distribution requires knowledge of the true event distribution. For images, networks trained a large corpus of image data are used to model these distributions, and metrics such as the Inception Score [132] or the Fréchet Inception Distance [62] are applied using these networks. However, this results in a second chicken and egg problem, as no such corpus of event data currently exists.

Instead, we evaluate our method directly on a set of downstream tasks, and demonstrate that our simulated events are able to train networks for complex tasks which generalize to data with real events. In Sections 8.4.1 and 8.4.2 we describe our experiments for 2D human pose estimation and object detection, respectively.

8.4.1 2D Human Pose Estimation

We train a 2D human pose detector for events based on the publicly available code from Xiao et al. [153], which uses an encoder-decoder style network to regress a heatmap for each desired joint. We use a ResNet-50 [58] encoder, pretrained on ImageNet [131]. For event inputs, we modify the number of input channels in the first layer, and randomly initialize the weights of this layer. The network is then trained on a 80%/20% split between the MPII [4] and Human3.6M [67] datasets. For each ground truth pose, the pair of images either 1 or 2 frames before and after the target frame are selected at random, and passed into the generator network to generate a simulated event volume.

We evaluate our method on the DHP19 [21] dataset, which consists of 3D joint positions of a human subject, recorded with motion capture, with events from four cameras surrounding the subject. Using the camera calibrations, we project these 3D joint positions into 2D image positions for each camera. Following the experiment schedule by Calabrese et al. [21], we use as a test set data from subjects 13-17 and cameras 2-3. As our method does not include any temporal consistency, we remove sequences with hand motions only, where most of the body is static and does not generate any events. This results in 16 motions across 5 subjects and 2 cameras. Following Calabrese et al. [21], we divide each sequence into chunks of 7500 events per camera, and evaluate on the average pose within each window.

One issue with this direct evaluation is that the marker positions for DHP19 vary significantly from those in MPII and H36M. In order to overcome this offset between the joint positions, we freeze all but the final linear layer of our network, and fine tune this layer on the DHP19 training set (subjects 1-12, cameras 2-3). This is equivalent to training a linear model on the activations from the second to last layer, as is common in the self-supervised learning literature [54].

8.4.2 Object Detection

We train a detection network using the YOLOv3 pipeline [125]. We initialize the network from a pretrained YOLOv3 network with spatial pyramid pooling, with the first input layer randomly initialized. The network is trained on simulated events from the KITTI Object Detection dataset [48], with the target frame and either the frame one or two frames prior.

	Pretrained only		1 Epoch		
	EventGAN	ESIM	EventGAN	ESIM	Real
MPJPE ↓	**14.55**	19.57	**6.76**	7.58	8.94
PCKh@50 ↑	**45.47**	40.53	**87.70**	85.89	80.55

	30 Epochs			140 epochs
	EventGAN	ESIM	Real	Real
MPJPE ↓	**6.44**	6.54	6.75	**6.39**
PCKh@50 ↑	**90.19**	89.93	87.53	89.86

Table 10: Human pose estimation results in MPJPE (pix.) (lower is better) and PCKh@50 (higher is better). All EventGAN and ESIM models are first pretrained on simulated events from the MPII and H36M datasets, and then the final linear layer is fine tuned on the DHP19 training set for the specified number of epochs. The Real models are trained directly (whole model) on the DHP19 training set for the specified number of epochs.

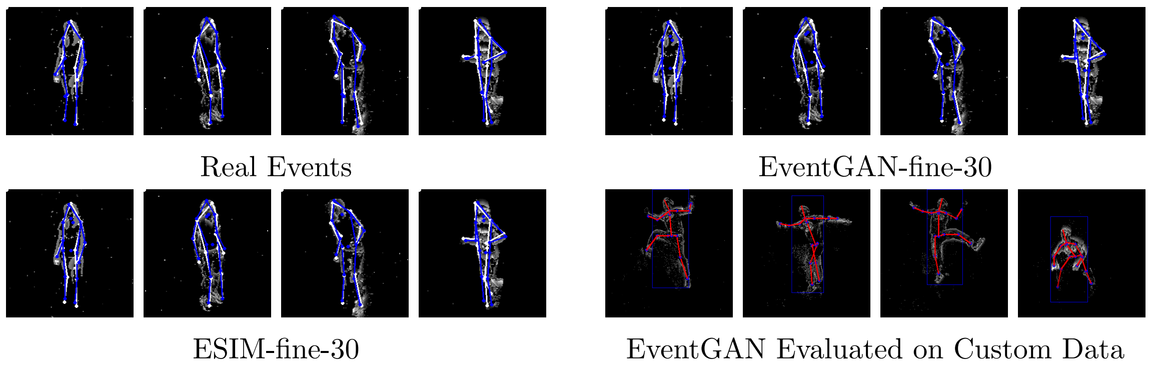


Figure 39: Qualitative results of our human pose estimation on real event data. The first three sets are evaluated on samples from the DHP19 dataset [21], where ground truth is in white and predictions are in blue. Our model is able to achieve accuracy on par with a model directly trained on the real data after 30 epoch of fine tuning only the last linear layer. The last set shows our YOLOv3 detection pipeline combined with our human pose estimator. The detection network is trained on MPII to detect the human in the scene (blue box), which is fed into the human pose estimator to estimate the 2D joint positions (MPII format). Best viewed in color.

8.4.3 The Event Car Detection Dataset

For evaluation, we generated a novel dataset for car bounding box annotations for event data. Our dataset consists of 250 labeled images from the MVSEC [171] outdoor driving dataset, with corresponding timestamps. For each image, raters label bounding boxes for all cars within the scene, while also separating the cars into easy (large, no occlusion), hard (medium, or partial occlusion) or don't care (mostly occluded or too small) categories. In total, there are 451 easy instances, 506 hard instances and 959 don't care instances. This dataset will be publicly available.

8.4.4 Competing Methods

We additionally simulate the MPII, H36M and KITTI datasets using ESIM [123], by simulating a random affine transform of each image in the dataset, similar to the method used by Rebecq et al. [124]. Using this simulated data, we train the same networks described in Sections 8.4.1 and 8.4.2. For both experiments, we also train

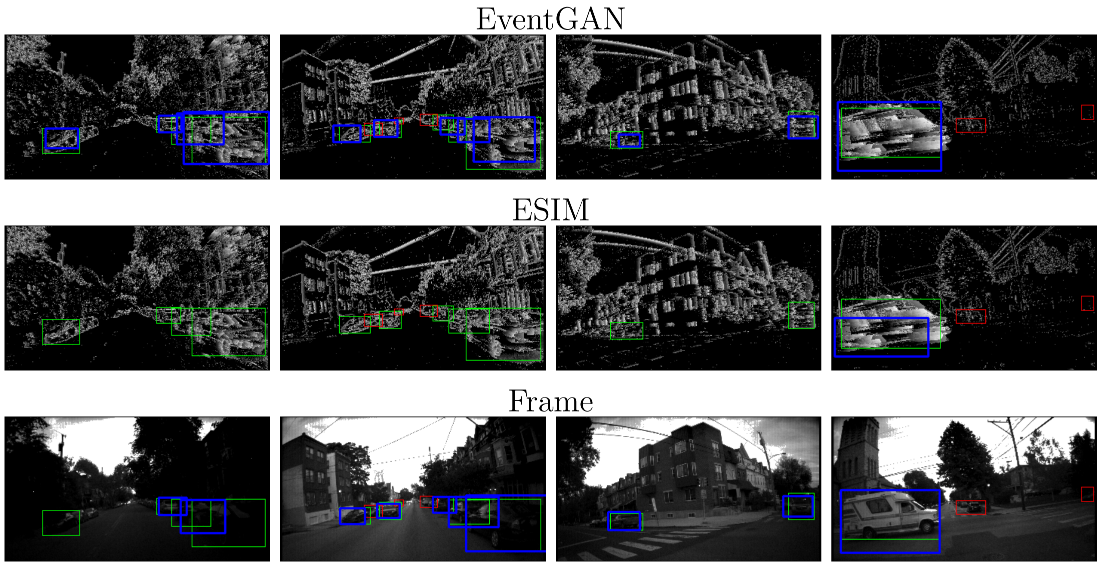


Figure 40: Selected qualitative results of our car detection pipeline using the YOLOv3 network [125]. Detections are in blue, GT labels in green, and don't care regions in red.

networks on real data as a baseline. For object detection, we train a network on the grayscale frames from KITTI, and evaluate on the grayscale frames from MVSEC and DDD17. For human pose estimation, we train a network on the events in the training set (subjects 1-12) of DHP19.

Training Data	Precision	Easy recall	Hard recall	Comb recall	AP	F-1
EventGAN	0.42	**0.57**	**0.34**	**0.45**	**0.30**	0.44
ESIM	0.23	0.08	0.02	0.05	0.02	0.09
Frame	**0.57**	0.48	0.27	0.37	0.29	**0.45**

Table 11: Object detection results on the Event Car Detection dataset. Metrics adopted from the PASCAL VOC challenge [37]. The EventGAN and ESIM models are trained on simulated events from the KITTI dataset, while the Frame model is trained on the real image frames from the KITTI dataset.

8.5 Results

8.5.1 2D Human Pose Estimation

We evaluate our method on the mean per joint position error (MPJPE) [21], $\frac{1}{N}\sum_i^N \|x_i - \hat{x}_i\|_2$, as well as PCKh@50 (percentage of correct keypoints) [4], which measures the percentage of joint predictions with error less than 50% of the head size. We define head size as $0.6\times$ the distance between the head and the midpoint between the shoulders.

In Table 10, we compare a network trained on simulated events from EventGAN, ESIM, and a network trained directly on the DHP19 training set. We also report results from fine tuning the final linear layer of the network on the DHP19 training set for both EventGAN and ESIM. Qualitative results from both DHP19 and out of sample data can be found in Figure 39. From these results, we can see that the data generated by EventGAN is able to train a network to learn representations that are very close to the true data. After only one epoch of fine tuning, and only of the final layer, we are able to achieve significantly higher accuracy than training on the real data, and come close to the accuracy of a network trained for 140 epochs on real data. However, the gap between ESIM and our method is also relatively small. This is largely due to the low difficulty of the dataset, as even training on real events converges to a relatively good solution after only one epoch of training. This was observed even when testing with much smaller networks, although they converge to a lower accuracy. The dataset is also much cleaner, and as such is closer to the ESIM outputs.

8.5.2 Object Detection

We evaluate our method according to the precision-recall statistics defined by the PASCAL VOC challenge [37]. Predictions with confidence < 0.2 are removed, and non-maximum suppression is applied for boxes with IoU > 0.2. In total, we report precision, recall on the easy and hard classes, as well as the AP and F-1 scores for each training input in Table 11. We provide qualitative results in Figure 40.

From these results, we observed that our method is able to achieve reasonably strong results, and comes close to matching the performance of the network with frame inputs, which was trained on real data. The difference in performance implies a small sim-to-real gap, but may also simply be due to a stronger signal in the images for certain frames (although this may also be true the other way round). On the other hand, the sim-to-real gap is significant when training on ESIM. As the true event distribution differs largely from the simulated data, the network is only able to perform accurate detections when the input has relatively low noise (e.g. Figure 40 right), resulting in very low recall.

8.6 Conclusions

In this work, we have proposed a novel method for training supervised neural networks for events using image data by way of image to event simulation. Given events and images from an event camera, our deep learning pipeline is able to accurately simulate events from a pair of grayscale images from existing image datasets. These events can be used to train downstream networks for complex tasks such as object detection and 2D human pose estimation, and generalize to real events.

The largest limitation of this work is the need for a pair of frames (video), thus prohibiting the use of larger image datasets such as ImageNet [131] and COCO [86]. While it is possible to train a GAN to predict events from a single image, this would become a complex future prediction task, as the GAN must hallucinate the motion within the image. Other promising future directions include exploring other event representations, more complicated adversarial architectures, and exploring more complex downstream tasks.

Chapter 9

Conclusions and Future Work

So far in this work, we have discussed several main issues facing event camera algorithm development, namely asynchronous processing, the lack of a photometric loss, the problem of space-time disentanglement, and the lack of training data. To resolve these issues, we have applied two main strategies in the previous chapters. Namely, they are to use the motion blur induced by projecting the events into the 2D image plane as a loss function, and applying unsupervised learning to bypass the need for large amounts of labeled training data. In addition, we have generated a large dataset with high quality ground truth, which we use for testing and evaluation only. This work has been targeted towards 3D perception tasks, such as visual odometry, optical flow and depth estimation. However, event cameras hold great potential beyond these tasks, as their high speed and high dynamic range properties can allow for robust classification type tasks beyond the scope of traditional cameras. In the final chapter, we have proposed a method to improve recognition for event cameras, by simulating events from images using a neural network, in order to take advantage of the large amount of labeled image data.

Moving forwards, there are a number of interesting research problems in this field. The most direct next step would be the integration of the proposed methods to physical robotic systems. While the benefits of these methods have been demonstrated on datasets, event cameras have yet to be tightly integrated into real-time robotics tasks such as closed loop control.

In terms of learning methods, temporal consistency seems to be a crucial factor in robust operation for event cameras in the real world. As the cameras only respond to changes in the scene, the only way to perceive static parts of the scene is through a memory of the past. Take the case, for example, of human pose estimation. When the person only moves their arms, the rest of the body does not generate any events. As a result, a network which is trained to predict the joints of the entire body, as in Section 8.4.1 would fail, unless it had been trained with a large corpus of arm only movements. As an event camera is essentially a motion detector, such a pipeline of tracking via detection seems like an inappropriate fit. Instead, a network which uses the events to update an internal state, for example of the joint positions, seems like a more appropriate output, as static joints would simply not be updated. For conventional neural networks, integrating a recursive neural network such as an LSTM [63] may be a promising direction.

However, conventional networks all face the problem of trying to fit a sparse, asynchronous sensing output into what is inherently a (usually) synchronous, dense processing pipeline. Spiking neural networks (SNNs) [51] are an alternative paradigm for learning which show great promise in taking full advantage of the sparse, asynchronous output from event cameras. By more closely mimicking the neurons in the brain, SNNs update the activations at each layer in an asynchronous fashion. This results in significantly lower bandwidth for sparse inputs, and much less communication required between layers. By developing SNNs for event cameras, we could have a fully asynchronous pipeline that goes from sensing to perception, which takes advantage of the low latency, high dynamic range, but also low power advantages of event cameras. While promising, the main issue currently surrounding SNNs is the challenge in training these networks. The main model used in present day SNNs is the Integrate and

Fire model [148], which accumulates input voltages at each node until it surpasses a given threshold, at which point a spike is sent to the next layer. This model is unfortunately non-differentiable at the transition point. Several methods [10, 136] have proposed surrogate gradients which model differentiable approximations to this model. However, these face issues similar to vanishing gradients for conventional neural networks which inhibits their use in deeper networks. Another promising direction in this field consists of local learning rules, such as Spike Timing Dependent Plasticity (STDP) [117], where the weights of the network are updated based only on the inputs of a neuron and its local neighbors. These methods do not require any gradient approximation, and are capable of unsupervised learning of distributions in the data. However, performing tasks beyond unsupervised learning with these methods is still an open problem.

Finally, a major constraint on the dissemination of event cameras into modern systems where conventional cameras thrive is the development of the silicon itself. For a while, event cameras were prohibitively expensive, pricing in the thousands per camera. In addition, camera resolution has been significantly lower than traditional cameras. As more companies begin to work on these cameras and develop hardware, we should hopefully see prices come down and resolution go up, resulting in adoption of event cameras in more real world applications.

LIST OF TABLES

LIST OF ILLUSTRATIONS

BIBLIOGRAPHY

[1] I. Alzugaray and M. Chli. Ace: An efficient asynchronous corner tracker for event cameras. In *2018 International Conference on 3D Vision (3DV)*, pages 653–661. IEEE, 2018.

[2] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7243–7252, 2017.

[3] A. Andreopoulos, H. J. Kashyap, T. K. Nayak, A. Amir, and M. D. Flickner. A low power, high throughput, fully event-based stereo system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7532–7542, 2018.

[4] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele. 2D Human pose estimation: New benchmark and state of the art analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[5] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.

[6] P. Bardow, A. J. Davison, and S. Leutenegger. Simultaneous optical flow and intensity estimation from an event camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 884–892, 2016.

[7] F. Barranco, C. Fermüller, and Y. Aloimonos. Contour motion estimation for asynchronous event-driven cameras. *Proceedings of the IEEE*, 102(10):1537–1556, 2014.

[8] F. Barranco, C. Fermuller, Y. Aloimonos, and T. Delbruck. A dataset for visual navigation with neuromorphic methods. *Frontiers in neuroscience*, 10, 2016.

[9] M. Bear, B. Connors, and M. Paradiso. *Neuroscience: Exploring the Brain.* Number v. 1 in Neuroscience: Exploring the Brain. Lippincott Williams & Wilkins, 2001. ISBN 9780683305968.

[10] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems*, pages 787–797, 2018.

[11] R. Benosman, S.-H. Ieng, P. Rogister, and C. Posch. Asynchronous event-based

Hebbian epipolar geometry. *IEEE Transactions on Neural Networks*, 22(11): 1723–1734, 2011.

[12] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. Srinivasan. Asynchronous frameless event-based optical flow. *Neural Networks*, 27:32–37, 2012.

[13] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi. Event-based visual flow. *IEEE Transactions on Neural Networks and Learning Systems*, 25 (2):407–417, 2014.

[14] F. Besse, C. Rother, A. Fitzgibbon, and J. Kautz. PMBP: Patchmatch belief propagation for correspondence field estimation. *International Journal of Computer Vision*, 110(1):2–13, 2014.

[15] Y. Bi and Y. Andreopoulos. PIX2NVS: Parameterized conversion of pixel-domain video frames to neuromorphic vision streams. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 1990–1994. IEEE, 2017.

[16] J. Binas, D. Neil, S. Liu, and T. Delbrück. DDD17: End-to-end DAVIS driving dataset. *CoRR*, abs/1711.01458, 2017.

[17] J. Binas, D. Niel, S.-C. Liu, and T. Delbruck. Ddd17: End-to-end davis driving dataset. 2017.

[18] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck. A 240$\times$ 180 130 db 3 μs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, 2014.

[19] T. Brosch, S. Tschechne, and H. Neumann. On event-based optical flow detection. *Frontiers in neuroscience*, 9:137, 2015.

[20] M. Z. Brown, D. Burschka, and G. D. Hager. Advances in computational stereo. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (8):993–1008, 2003.

[21] E. Calabrese, G. Taverni, C. Awai Easthope, S. Skriabine, F. Corradi, L. Longinotti, K. Eng, and T. Delbruck. DHP19: Dynamic vision sensor 3D human pose dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[22] L. A. Camuñas-Mesa, T. Serrano-Gotarredona, S. H. Ieng, R. B. Benosman, and B. Linares-Barranco. On the use of orientation filters for 3D reconstruction in event-driven stereo vision. *Frontiers in Neuroscience*, 8, 2014.

[23] L. A. Camuñas-Mesa, T. Serrano-Gotarredona, S.-H. Ieng, R. Benosman, and B. Linares-Barranco. Event-driven stereo visual tracking algorithm to solve object occlusion. *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[24] J. Canny. A computational approach to edge detection. In *Readings in computer vision*, pages 184–203. Elsevier, 1987.

[25] J. Carneiro, S.-H. Ieng, C. Posch, and R. Benosman. Event-based 3D reconstruction from neuromorphic retinas. *Neural Networks*, 45:27–38, 2013.

[26] A. Censi and D. Scaramuzza. Low-latency event-based visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 703–710. IEEE, 2014.

[27] A. Censi, J. Strubel, C. Brandli, T. Delbruck, and D. Scaramuzza. Low-latency localization by active led markers tracking using a dynamic vision sensor. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 891–898. IEEE, 2013.

[28] K. Chaney, A. Zihao Zhu, and K. Daniilidis. Learning event-based height from plane and parallax. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[29] P. Charbonnier, L. Blanc-Féraud, G. Aubert, and M. Barlaud. Two deterministic half-quadratic regularization algorithms for computed imaging. In *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, volume 2, pages 168–172. IEEE, 1994.

[30] L. Clement, V. Peretroukhin, J. Lambert, and J. Kelly. The battle for filter supremacy: A comparative study of the multi-state constraint kalman filter and the sliding window filter. In *Computer and Robot Vision (CRV), 2015 12th Conference on*, pages 23–30, 2015.

[31] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 142–149. IEEE, 2000.

[32] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE Computer Society, 2005.

[33] K. Daniilidis. Hand-eye calibration using dual quaternions. *The International Journal of Robotics Research*, 18(3):286–298, 1999.

[34] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, et al. Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[35] E. H. de Haan and A. Cowey. On the usefulness of whatand wherepathways in vision. *Trends in cognitive sciences*, 15(10):460–466, 2011.

[36] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):611–625, 2018.

[37] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[38] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54, 2006.

[39] M. Firouzi and J. Conradt. Asynchronous event-based cooperative stereo matching using neuromorphic silicon retinas. *Neural Processing Letters*, 43 (2):311–326, 2016. ISSN 1573-773X. doi: 10.1007/s11063-015-9434-5. URL `http://dx.doi.org/10.1007/s11063-015-9434-5`.

[40] P. Furgale, T. D. Barfoot, and G. Sibley. Continuous-time batch estimation using temporal basis functions. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2088–2095. IEEE, 2012.

[41] P. Furgale, J. Rehder, and R. Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1280–1286, 2013. doi: 10.1109/IROS.2013.6696514.

[42] P. Furgale, J. Rehder, and R. Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1280–1286. IEEE, 2013.

[43] G. Gallego and D. Scaramuzza. Accurate angular velocity estimation with an event camera. *IEEE Robotics and Automation Letters*, 2(2):632–639, 2017.

[44] G. Gallego, J. E. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza. Event-based, 6-dof camera tracking for high-speed applications. *arXiv preprint arXiv:1607.03468*, 2016.

[45] G. Gallego, H. Rebecq, and D. Scaramuzza. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical

flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3867–3876, 2018.

[46] D. Gehrig, H. Rebecq, G. Gallego, and D. Scaramuzza. Asynchronous, photometric feature tracking using events and frames. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 750–765, 2018.

[47] A. Geiger, M. Roser, and R. Urtasun. Efficient large-scale stereo matching. In *Asian conference on computer vision*, pages 25–38. Springer, 2010.

[48] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[49] A. Geiger, F. Moosmann, mer Car, and B. Schuster. Automatic camera and range sensor calibration using a single shot. In *International Conference on Robotics and Automation (ICRA)*, St. Paul, USA, May 2012.

[50] R. Ghosh, A. Mishra, G. Orchard, and N. V. Thakor. Real-time object recognition and orientation estimation using an event-based camera and CNN. In *Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE*, pages 544–547. IEEE, 2014.

[51] S. Ghosh-Dastidar and H. Adeli. Spiking neural networks. *International journal of neural systems*, 19(04):295–308, 2009.

[52] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *CVPR*, volume 2, page 7, 2017.

[53] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[54] P. Goyal, D. Mahajan, A. Gupta, and I. Misra. Scaling and benchmarking self-supervised visual representation learning. *arXiv preprint arXiv:1905.01235*, 2019.

[55] S. Granger and X. Pennec. Multi-scale em-icp: A fast and robust approach for surface registration. In *European Conference on Computer Vision*, pages 418–432. Springer, 2002.

[56] G. Haessig, A. Cassidy, R. Alvarez, R. Benosman, and G. Orchard. Spiking optical flow for event-based sensors using ibm's truenorth neurosynaptic system. *IEEE transactions on biomedical circuits and systems*, 12(4):860–870, 2018.

[57] C. Harris and M. Stephens. A combined corner and edge detector. Citeseer, 1988.

[58] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[59] L. Heng, B. Li, and M. Pollefeys. Camodocal: Automatic intrinsic and extrinsic calibration of a rig with multiple generic cameras and odometry. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1793–1800. IEEE, 2013.

[60] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis. Observability-constrained vision-aided inertial navigation. *University of Minnesota, Dept. of Comp. Sci. & Eng., MARS Lab, Tech. Rep*, 1, 2012.

[61] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2D LIDAR SLAM. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1271–1278. IEEE, 2016.

[62] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[63] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[64] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[65] Y. Hu, H. Liu, M. Pfeiffer, and T. Delbruck. DVS benchmark datasets for object tracking, action recognition, and object recognition. *Frontiers in neuroscience*, 10, 2016.

[66] M. Iacono, S. Weber, A. Glover, and C. Bartolozzi. Towards event-driven object detection with off-the-shelf deep learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[67] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu. Human3.6M: Large scale datasets and predictive methods for 3D human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339, jul 2014.

[68] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with

conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[69] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[70] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[71] J. Kaiser, J. C. V. Tieck, C. Hubschneider, P. Wolf, M. Weber, M. Hoff, A. Friedrich, K. Wojtasik, A. Roennau, R. Kohlhaas, et al. 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)*, pages 127–134. IEEE, 2016.

[72] H. Kim, A. Handa, R. Benosman, S.-H. Ieng, and A. J. Davison. Simultaneous mosaicing and tracking with an event camera. *J. Solid State Circ*, 43:566–576, 2008.

[73] H. Kim, A. Handa, R. Benosman, S.-H. Ieng, and A. J. Davison. Simultaneous mosaicing and tracking with an event camera. In *British Machine Vision Conference*. IEEE, 2014.

[74] H. Kim, S. Leutenegger, and A. J. Davison. Real-time 3d reconstruction and 6-dof tracking with an event camera. In *European Conference on Computer Vision*, pages 349–364. Springer, 2016.

[75] J. Kogler, C. Sulzbachner, F. Eibensteiner, and M. Humenberger. Address-event matching for a silicon retina based stereo vision system. In *4th Int. Conference from Scientific Computing to Computational Engineering*, pages 17–24, 2010.

[76] J. Kogler, M. Humenberger, and C. Sulzbachner. Event-based stereo matching approaches for frameless address event stereo data. *Advances in Visual Computing*, pages 674–685, 2011.

[77] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza. Low-latency visual odometry using event-based feature tracks. In *IEEE/RSJ International Conference on Robotics and Intelligent Systems*. IEEE/RSJ, 2016.

[78] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza. Low-latency visual odometry using event-based feature tracks. In *IEEE/RSJ International Confer-*

ence on Intelligent Robots and Systems (IROS), number EPFL-CONF-220001, 2016.

[79] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat, and R. Benosman. Asynchronous event-based multikernel algorithm for high-speed visual features tracking. *IEEE transactions on neural networks and learning systems*, 26(8):1710–1720, 2015.

[80] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat, and R. Benosman. Asynchronous event-based multikernel algorithm for high-speed visual features tracking. *IEEE transactions on neural networks and learning systems*, 26(8):1710–1720, 2015.

[81] 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*, 39(7):1346–1359, 2017.

[82] W.-S. Lai, J.-B. Huang, and M.-H. Yang. Semi-supervised learning for optical flow with generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 353–363, 2017.

[83] 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 *29th British Machine Vision Conference 2018*, 2018.

[84] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128×128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE journal of solid-state circuits*, 43(2):566–576, 2008.

[85] J. H. Lim and J. C. Ye. Geometric gan. *arXiv preprint arXiv:1705.02894*, 2017.

[86] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[87] M. Litzenberger, C. Posch, D. Bauer, A. Belbachir, P. Schon, B. Kohn, and H. Garn. Embedded vision system for real-time object tracking using an asynchronous transient vision sensor. In *2006 IEEE 12th Digital Signal Processing Workshop & 4th IEEE Signal Processing Education Workshop*, pages 173–178. IEEE, 2006.

[88] M. Liu and T. Delbruck. Abmof: A novel optical flow algorithm for dynamic vision sensors. *arXiv preprint arXiv:1805.03988*, 2018.

[89] B. Lucas and T. Kanade. An iterative image registration technique with an ap-

plication to stereo vision. In *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, volume 81, pages 674–679, 1981.

[90] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5419–5427, 2018.

[91] D. Marr, T. Poggio, et al. Cooperative computation of stereo disparity. *From the Retina to the Neocortex*, pages 239–243, 1976.

[92] J. Maye, P. Furgale, and R. Siegwart. Self-supervised calibration for robotic systems. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 473–480. IEEE, 2013.

[93] S. Meister, J. Hur, and S. Roth. Unflow: Unsupervised learning of optical flow with a bidirectional census loss. *arXiv preprint arXiv:1711.07837*, 2017.

[94] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[95] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[96] 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)*, pages 1–9. IEEE, 2018.

[97] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbrück. Steering a predator robot using a mixed frame/event-driven convolutional neural network. In *Event-based Control, Communication, and Signal Processing (EBCCSP), 2016 Second International Conference on*, pages 1–8. IEEE, 2016.

[98] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. 2006.

[99] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. Technical report, 2007.

[100] 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.

[101] E. Mueggler, C. Forster, N. Baumli, G. Gallego, and D. Scaramuzza. Lifetime estimation of events from dynamic vision sensors. In *2015 IEEE international conference on Robotics and Automation (ICRA)*, pages 4874–4881. IEEE, 2015.

[102] E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza. Continuous-time visual-inertial trajectory estimation with event cameras. *arXiv preprint arXiv:1702.07389*, 2017.

[103] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *The International Journal of Robotics Research*, 36 (2):142–149, 2017.

[104] E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza. Continuous-time visual-inertial odometry for event cameras. *IEEE Transactions on Robotics*, (99):1–16, 2018.

[105] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*, pages 483–499. Springer, 2016.

[106] A. Nguyen, T.-T. Do, D. G. Caldwell, and N. G. Tsagarakis. Real-time pose estimation for event cameras with stacked spatial lstm networks. *arXiv preprint arXiv:1708.09011*, 2017.

[107] Z. Ni, S.-H. Ieng, C. Posch, S. Régnier, and R. Benosman. Visual tracking using neuromorphic asynchronous event-based cameras. *Neural computation*, 2015.

[108] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart. A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 431–437, May 2014. doi: 10.1109/ICRA.2014.6906892.

[109] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3400–3407. IEEE, 2011.

[110] G. Orchard and R. Etienne-Cummings. Bioinspired visual motion estimation. *Proceedings of the IEEE*, 102(10):1520–1536, 2014.

[111] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9, 2015.

[112] P. K. Park, B. H. Cho, J. M. Park, K. Lee, H. Y. Kim, H. A. Kang, H. G. Lee, J. Woo, Y. Roh, W. J. Lee, et al. Performance improvement of deep learning based gesture recognition using spatiotemporal demosaicing technique. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 1624–1628. IEEE, 2016.

[113] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[114] E. Piatkowska, A. N. Belbachir, and M. Gelautz. Cooperative and asynchronous stereo vision for dynamic vision sensors. *Measurement Science and Technology*, 25(5):055108, 2014.

[115] E. Piatkowska, J. Kogler, N. Belbachir, and M. Gelautz. Improved cooperative stereo matching for dynamic vision sensors with ground truth evaluation. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 370–377. IEEE, 2017.

[116] C. Posch, D. Matolin, and R. Wohlgenannt. An asynchronous time-based image sensor. In *2008 IEEE International Symposium on Circuits and Systems*, pages 2130–2133. IEEE, 2008.

[117] R. P. Rao and T. J. Sejnowski. Spike-timing-dependent hebbian plasticity as temporal difference learning. *Neural computation*, 13(10):2221–2237, 2001.

[118] H. Rebecq, G. Gallego, E. Mueggler, and D. Scaramuzza. Emvs: Event-based multi-view stereo3d reconstruction with an event camera in real-time. *International Journal of Computer Vision*, pages 1–21, 2017.

[119] H. Rebecq, T. Horstschaefer, G. Gallego, and D. Scaramuzza. EVO: A geometric approach to event-based 6-dof parallel tracking and mapping in real time. *IEEE Robotics and Automation Letters*, 2(2):593–600, 2017.

[120] H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. In *BMVC*, 2017.

[121] H. Rebecq, T. Horstschaefer, and D. Scaramuzza. Real-time visual-inertial

odometry for event cameras using keyframe-based nonlinear optimization. In *British Machine Vis. Conf.(BMVC)*, volume 3, 2017.

[122] H. Rebecq, G. Gallego, E. Mueggler, and D. Scaramuzza. Emvs: Event-based multi-view stereo3d reconstruction with an event camera in real-time. *International Journal of Computer Vision*, 126(12):1394–1414, 2018.

[123] H. Rebecq, D. Gehrig, and D. Scaramuzza. Esim: an open event camera simulator. In *Conference on Robot Learning*, pages 969–982, 2018.

[124] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza. Events-to-video: Bringing modern computer vision to event cameras. *arXiv preprint arXiv:1904.08298*, 2019.

[125] J. Redmon and A. Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[126] C. Reinbacher, G. Graber, and T. Pock. Real-time intensity-image reconstruction for event cameras using manifold regularisation. In *British Machine Vision Conference*, 2016.

[127] Z. Ren, J. Yan, B. Ni, B. Liu, X. Yang, and H. Zha. Unsupervised deep learning for optical flow estimation. In *AAAI*, pages 1495–1501, 2017.

[128] P. Rogister, R. Benosman, S.-H. Ieng, P. Lichtsteiner, and T. Delbruck. Asynchronous event-based binocular stereo matching. *IEEE Transactions on Neural Networks and Learning Systems*, 23(2):347–353, 2012.

[129] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[130] B. Rueckauer and T. Delbruck. Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor. *Frontiers in neuroscience*, 10, 2016.

[131] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[132] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[133] S. Schraml, A. Nabil Belbachir, and H. Bischof. Event-driven stereo matching for real-time 3D panoramic vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 466–474, 2015.

[134] J. Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[135] J. Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[136] S. B. Shrestha and G. Orchard. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, pages 1412–1421, 2018.

[137] F. Stein. Efficient computation of optical flow using the census transform. In *Joint Pattern Recognition Symposium*, pages 79–86. Springer, 2004.

[138] D. Sun, S. Roth, and M. J. Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

[139] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.

[140] D. Tedaldi, G. Gallego, E. Mueggler, and D. Scaramuzza. Feature detection and tracking with the dynamic and active-pixel vision sensor (davis). In *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pages 1–7. IEEE, 2016.

[141] D. Tran, R. Ranganath, and D. Blei. Hierarchical implicit models and likelihood-free variational inference. In *Advances in Neural Information Processing Systems*, pages 5523–5533, 2017.

[142] C. Troiani, A. Martinelli, C. Laugier, and D. Scaramuzza. 2-point-based outlier rejection for camera-imu systems with applications to micro aerial vehicles. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5530–5536. IEEE, 2014.

[143] S. Tschechne, T. Brosch, R. Sailer, N. von Egloffstein, L. I. Abdul-Kreem, and H. Neumann. On event-based motion detection and integration. In *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies*, BICT '14, 2014.

[144] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. 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.

[145] S. Vijayanarasimhan, S. Ricco, C. Schmid, R. Sukthankar, and K. Fragkiadaki. Sfm-net: Learning of structure and motion from video. *arXiv preprint arXiv:1704.07804*, 2017.

[146] R. Wang, J.-M. Frahm, and S. M. Pizer. Recurrent neural network for learning densedepth and ego-motion from video. *arXiv preprint arXiv:1805.06558*, 2018.

[147] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger. Pseudo-LiDAR from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8445–8453, 2019.

[148] U. Wehmeier, D. Dong, C. Koch, and D. Van Essen. Modeling the mammalian visual system. In *Methods in neuronal modeling*, pages 335–359. MIT Press, 1989.

[149] D. Weikersdorfer and J. Conradt. Event-based particle filtering for robot self-localization. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 866–870. IEEE, 2012.

[150] D. Weikersdorfer, R. Hoffmann, and J. Conradt. Simultaneous localization and mapping for event-based vision systems. In *International Conference on Computer Vision Systems*, pages 133–142. Springer Berlin Heidelberg, 2013.

[151] D. Weikersdorfer, D. B. Adrian, D. Cremers, and J. Conradt. Event-based 3d slam with a depth-augmented dynamic vision sensor. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 359–364, 2014.

[152] X. Weng and K. Kitani. Monocular 3D object detection with pseudo-LiDAR point cloud. *arXiv preprint arXiv:1903.09847*, 2019.

[153] B. Xiao, H. Wu, and Y. Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 466–481, 2018.

[154] Z. Xie, S. Chen, and G. Orchard. Event-based stereo depth estimation using belief propagation. *Frontiers in Neuroscience*, 11:535, 2017.

[155] Z. Xie, J. Zhang, and P. Wang. Event-based stereo matching using

semiglobal matching. *International Journal of Advanced Robotic Systems*, 15 (1):1729881417752759, 2018.

[156] C. Ye, A. Mitrokhin, C. Parameshwara, C. Fermüller, J. A. Yorke, and Y. Aloimonos. Unsupervised learning of dense optical flow and depth from sparse event data. *arXiv preprint arXiv:1809.08625*, 2018.

[157] Y. You, Y. Wang, W.-L. Chao, D. Garg, G. Pleiss, B. Hariharan, M. Campbell, and K. Q. Weinberger. Pseudo-LiDAR++: Accurate depth for 3d object detection in autonomous driving. *arXiv preprint arXiv:1906.06310*, 2019.

[158] J. J. Yu, A. W. Harley, and K. G. Derpanis. Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness. In *Computer Vision–ECCV 2016 Workshops*, pages 3–10. Springer, 2016.

[159] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *European conference on computer vision*, pages 151–158. Springer, 1994.

[160] A. Zanardi, A. Aumiller, J. Zilly, A. Censi, and E. Frazzoli. Cross-modal learning filters for rgb-neuromorphic wormhole learning. *Robotics: Science and System XV*, page P45, 2019.

[161] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[162] H. Zhan, R. Garg, C. S. Weerasekera, K. Li, H. Agarwal, and I. Reid. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 340–349, 2018.

[163] J. Zhang and S. Singh. LOAM: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, 2014.

[164] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, volume 2, page 7, 2017.

[165] Y. Zhou, G. Gallego, H. Rebecq, L. Kneip, H. Li, and D. Scaramuzza. Semi-dense 3d reconstruction with a stereo event camera. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 235–251, 2018.

[166] A. Z. Zhu, N. Atanasov, and K. Daniilidis. Event-based feature tracking

with probabilistic data association. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.

[167] A. Z. Zhu, N. Atanasov, and K. Daniilidis. Event-based feature tracking with probabilistic data association. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 4465–4470. IEEE, 2017.

[168] A. Z. Zhu, N. Atanasov, and K. Daniilidis. Event-based visual inertial odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5391–5399, 2017.

[169] A. Z. Zhu, N. Atanasov, and K. Daniilidis. Event-based visual inertial odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5391–5399, 2017.

[170] A. Z. Zhu, D. Thakur, T. Ozaslan, B. Pfrommer, V. Kumar, and K. Daniilidis. The multi vehicle stereo event camera dataset: An event camera dataset for 3D perception. *IEEE Robotics and Automation Letters*, 2018.

[171] A. Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, and K. Daniilidis. The multivehicle stereo event camera dataset: An event camera dataset for 3D perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, 2018.

[172] A. Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, and K. Daniilidis. The multi vehicle stereo event camera dataset: An event camera dataset for 3D perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, 2018.

[173] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis. EV-Flownet: Self-supervised optical flow estimation for event-based cameras. *arXiv preprint arXiv:1802.06898*, 2018.

[174] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 989–997, 2019.

[175] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[176] Y. Zhu, Z. Lan, S. Newsam, and A. G. Hauptmann. Guided optical flow learning. *arXiv preprint arXiv:1702.02295*, 2017.

[177] T. Zinßer, J. Schmidt, and H. Niemann. Point set registration with integrated

scale estimation. In *Eighth International Conference on Pattern Recognition and Image Processing*, volume 116, page 119, 2005.

[178] D. Zou, P. Guo, Q. Wang, X. Wang, G. Shao, F. Shi, J. Li, and P.-K. Park. Context-aware event-driven stereo matching. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 1076–1080. IEEE, 2016.

[179] D. Zou, F. Shi, W. Liu, J. Li, Q. Wang, P.-K. Park, C.-W. Shi, Y. J. Roh, and H. E. Ryu. Robust dense depth map estimation from sparse DVS stereos. In *British Machine Vis. Conf.(BMVC)*, volume 3, 2017.