

**דו"ח סיכום פרויקט: ב'**

**ניקוי רעשים במצלמת אירועים**

**Event Camera Denoising**

**מבצעים: By:**

בר וייס Bar Weiss

אסף עומר Asaf Omer

**מנחה:Supervisor:**

יונתן שדמי Yonatan Shadmi

**סמסטר רישום: אביב תשפ"ב**

**תאריך הגשה: ‏פברואר, 2023**



**בשיתוף עם: רפאל**

Table of Contents

[Abstract 4](#_Toc126274871)

[1 Introduction 4](#_Toc126274872)

[1.1 Introduction to Event Cameras 4](#_Toc126274873)

[1.2 Advantages and Applications of Event Cameras 5](#_Toc126274874)

[1.3 Event Camera Circuit Operation 6](#_Toc126274875)

[1.4 Event Generation Model 8](#_Toc126274876)

[1.5 V2E Tool 9](#_Toc126274877)

[2 Noise in Event Cameras 10](#_Toc126274878)

[​2.1 Background Activity 10](#_Toc126274879)

[2.2 Shot Noise 11](#_Toc126274880)

[2.3 Hot / Cold Pixels 11](#_Toc126274881)

[2.4 Threshold Mismatch 12](#_Toc126274882)

[2.5 Threshold Mismatch Event Generation Model 13](#_Toc126274883)

[3. Simulation Environment 15](#_Toc126274884)

[3.1 Implementing a Background Activity Filter 15](#_Toc126274885)

[3.2 Implementing a Video-to-Event Simulation Environment 17](#_Toc126274886)

[4. Threshold Correction 20](#_Toc126274887)

[4.1 Threshold Estimation Algorithm 20](#_Toc126274888)

[4.2 Threshold Correction Algorithm 22](#_Toc126274889)

[4.3 Incorporation in General Filtering Scheme 24](#_Toc126274890)

[5. Performance Analysis 25](#_Toc126274891)

[5.1 Threshold Estimation Performance 25](#_Toc126274892)

[5.2 Threshold Correction Performance with Perfect Estimation 27](#_Toc126274893)

[5.3 Threshold Correction Performance with Estimation Error 28](#_Toc126274894)

[5. Conclusion 30](#_Toc126274895)

[7. Further Work 30](#_Toc126274896)

[References 31](#_Toc126274897)

# Abstract

In this work we explore the different noise mechanisms in event cameras and discuss known methods for filtering several noise sources. We focus on the threshold mismatch effect in event cameras and introduce a novel correction scheme to reduce the effect of threshold mismatch. To the best of our knowledge this is the first algorithmic solution to the threshold mismatch effect. We use a threshold estimation algorithm suggested by Ziwei Wang et al [7] and further explore it in simulations with knowledge of the thresholds to calculate estimation errors. Using the estimated thresholds, we apply a correction algorithm based on sampling theory. These methods are tested and evaluated in simulations to provide reference to ground truth for performance evaluation. We also provide insight on performance evaluation and suggest and apply a metric for pixel uniformity.

# 1 Introduction

## 1.1 Introduction to Event Cameras

Event Cameras, sometimes referred to as a neuromorphic camera, or a Dynamic Vision Sensor (DVS), are asynchronous image sensors that respond to changes in brightness levels. Unlike traditional cameras, in which an external clock dictates the rate of capture for image acquisition, pixels of an event camera behave independently. Each pixel of a DVS will generate “Events”, which are created when the brightness level (log intensity) detected by the pixel’s photodiode, crosses a pre-defined threshold. This threshold is a reference brightness level that is stored in each individual pixel. Two types of events can be generated in this way: positive and negative polarity events (also named ON and OFF events respectively). The polarity is determined by the type of change in brightness levels - increase (positive polarity event) or decrease (negative polarity event). In most event cameras, the threshold is set by the user, is uniform across pixels (though stored individually in each), and is not strictly symmetrical (i.e., the positive threshold is not necessarily equal to the absolute value of the negative threshold).

Figure 1: DAVIS346 Event Camera [8]

Figure DAVIS346 camera [8]

This mode of operation was inspired by biological visual pathways, notably in the human visual system, where neurons carrying information through the visual pathways report spikes in the brightness level we sense, and are asynchronous and independent, like the pixels of a DVS. In the human visual system neurons pass information through a stream of identical pulses, the information is contained in the frequency of these pulses since they are all identical these pulses correlates to the stimulation of the neuron. Similarly, in event cameras information is passed with 2 types of pulses (negative and positive) and the scene's brightness changes are translated to the time interval between events/pulses. A fast brightness change will cause a short time interval, and vice versa a slow brightness change will cause a longer time interval between events.

The output of the camera is not a set of frames, but rather a stream of data called AER [2] (Address Event Representation). Each event is represented with 4 numbers . is called the polarity of the event and represents if it was negative or positive. contains the x-y pixel coordinates of the event. stores the timestamp of the event. Some event cameras will output additional information, such as a measurement of brightness level at the time of an event or at a constant frame rate such as the DAVIS (Dynamic and Active Pixel Vision Sensor) ,or IMU (inertial measurement unit) data that indicates change in camera position between events [1].

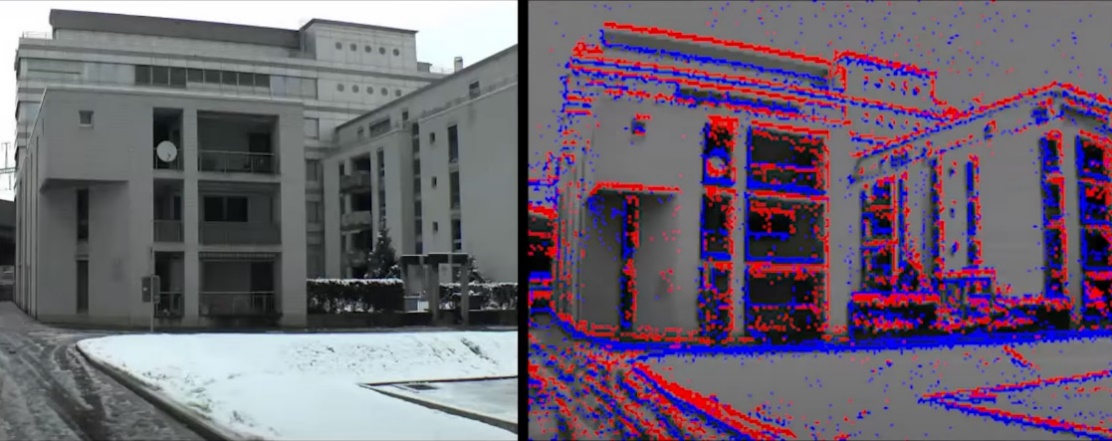


Figure Event Camera Output Visualization [9]

Event cameras, unlike traditional sensors, emphasize the dynamics of the captured scene. This is because the incident light on each pixel is a product of two factors (1) single object illumination change, (2) object movement. In most natural scenes, single objects' illumination is either constant or varies slowly overtime. This means that when the brightness level (log intensity) changes beyond a threshold for some pixel, the change is most likely due to movement of objects across the camera’s field of view. This makes event cameras especially effective in detecting motion by ignoring static objects since their illumination is constant.

## 1.2 Advantages and Applications of Event Cameras

Due to the independent nature of event generation in pixels, the temporal frequency of an event camera is not limited by a set frame rate. Rather, the acquisition rate is dependent on the scene dynamics. This carries two main advantages; (1) Very high temporal resolution with low latency, unlimited by an external clock, and (2) low power and memory consumption, as static scenes (or static portions of all scenes) will generate no events. Additionally, the dynamic range of DVS is significantly higher than that of standard cameras (140 dB vs. 60 dB) [1].

While a relatively novel technology (commercially available only since 2008), DVS sensors already see use in many fields. The dynamics-based operation mode of event cameras makes them valuable in object tracking, surveillance, and monitoring. In robotics, their low power consumption and effectiveness in low lighting conditions are a key advantage. Their high temporal resolution is valuable for gesture recognition and object detection. Additionally, the DVS can work simultaneously with traditional sensors to allow image deblurring and other high-level forms of filtering.



Figure Event Camera Surveillance Application [10]

Some disadvantages of event cameras include:

1. Storing and processing event data, especially in real time, can be computationally intensive and slow.
2. Event cameras cannot replace traditional sensors completely as they omit information like the color of the scene, static objects, and background.
3. Events are transmitted using a shared digital output bus, which can become saturated and slow the rate events are being sent.
4. Event cameras are still susceptible to different types of noise, some of which are experienced in traditional sensors, and others that are unique to DVS technology.

## 1.3 Event Camera Circuit Operation

As discussed above, each pixel of a standard event camera operates independently of the others. This means that in addition to a photoreceptor, each pixel must also include a circuit that can store the reference threshold voltage, and compare the measured brightness level to it, generating events.

The most basic and common implementation of this circuit is represented in figure 4. Light is detected by a photodiode, generating photocurrent . This generated current is fed into a feedback diode, which sets a voltage value with a logarithmic proportion to the light intensity (other implementations use an inverting amplifier in combination with a PMOS transistor). The reset level is set to the voltage that last generated an event . Therefore, the capacitor holds the voltage change which is proportional to the brightness change that we want to measure. The voltage is then inverted and amplified to . is fed into a differencing circuit containing two comparators, with reference voltages (thresholds) and , which will generate a positive or negative event respectively, given that the measured voltage surpasses the reference voltage. Typically, threshold values are . Once an event is generated, the new log intensity value is stored at the reset level, such that future events will only be generated relatively to the illumination in the last event.

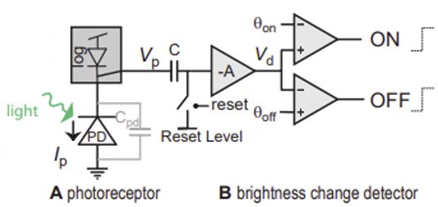


Figure Basic Pixel Circuit Implementation, figure taken from [3]

Figure 5 shows the operating principle of an event camera. Given an exponential increase in brightness level, a linear increase in is generated. This will continuously create positive polarity events, upon reaching the threshold, each time returning to reset level when an event is generated. As the brightness level drops at a slower rate, off events are generated at sporadic points. The capacitor allows accumulating small changes in brightness overtime until enough of a shift is detected to generate an event.

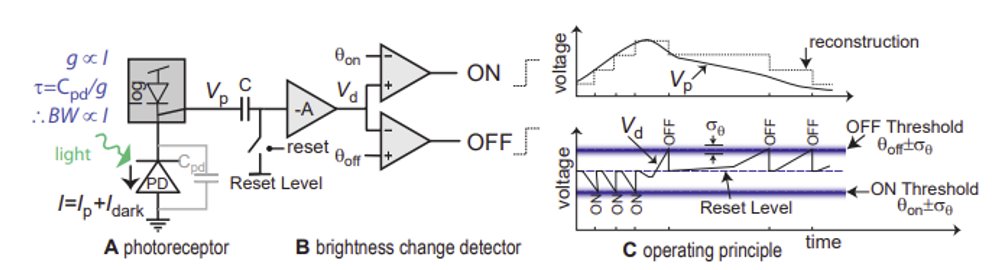


Figure Event Generation Operating Principle, figure taken from [3]

Additionally, note that the threshold level is not represented as a singular value, as threshold mismatch causes the / thresholds to vary with standard deviation , . This phenomenon will be explored further in later sections of the report.

## 1.4 Event Generation Model

As we saw in the previous chapters event cameras are sensitive to log changes of the diode photocurrent, here we would like to suggest a simple mathematical model for how events are created. We denote the log photocurrent as , and define for an event at pixel the log difference from the last event as [1]

Where is the time passed from the last event at pixel . When an event occurs, it means that the change has surpassed the threshold depending on the event's polarity, therefore

In principle the threshold is set to be uniform for all pixels, but in practice they will be different for each pixel and even the ON and OFF thresholds may be different, that phenomenon is called threshold mismatch and will be elaborated on in later chapters. This will change the model slightly.

## 1.5 V2E Tool

V2e [3] is a tool developed by Yuhuang Hu et al that allows us to create simulated event data from standard Visible Imaging System (VIS) camera inputs. The tool is quite complicated because it covers many different effects in event cameras, a lot of them are not very relevant to our discussion so we will not cover them all in this document. Here we will explain the general pipeline of the tool and the modeling of relevant effects.

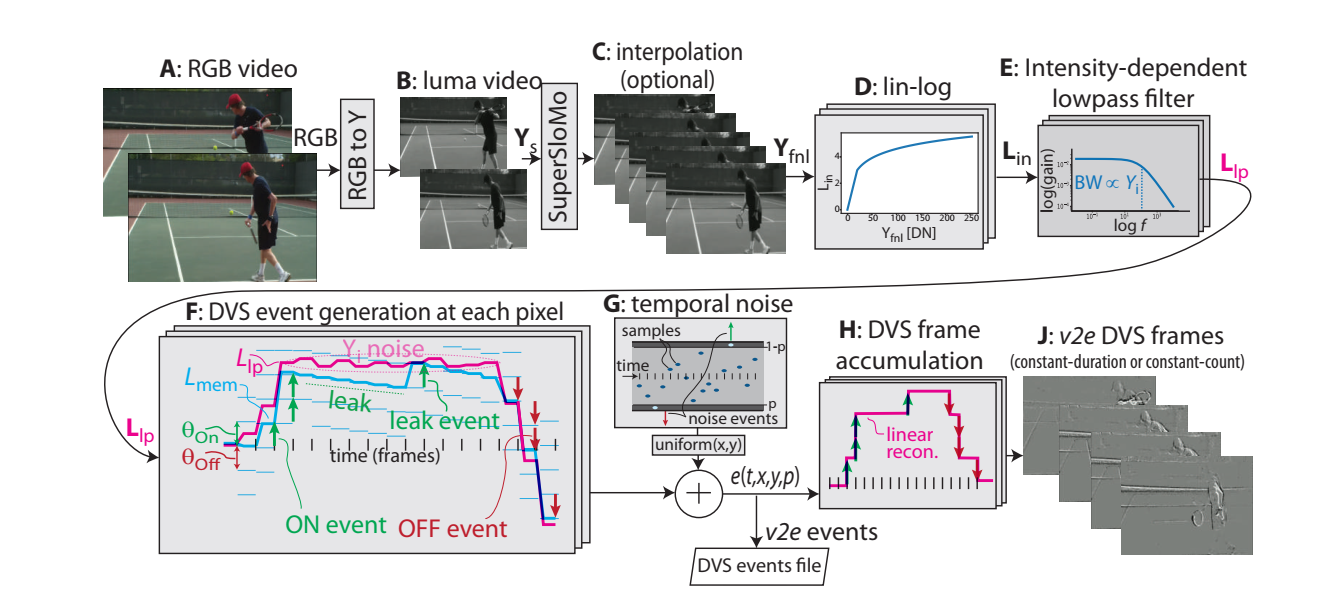


Figure steps of the v2e DVS event generation, figure taken from [3]

The stages of the algorithm are demonstrated in figure 6. **A-B:** first we convert the RGB video to luma values since the camera is sensitive to intensity and doesn’t distinguish between colors. **B-C:** to mimic the high temporal resolution of DVS the video is interpolated to a higher frame rate using a neural network called Super-SloMo. That allows us to detect changes over temporal intervals that are shorter than the original frame rate bringing us closer to the DVS temporal resolution. **C-D:** as we stated before the DVS is sensitive to log-intensity changes rather than linear changes so the intensity values are converted using a lin-log function, which is simply linear for inputs and logarithmic for and is continuous. In other words it is defined as

This is done to decrease the quantization error for small intensity values. **E:** the DVS pixel circuit has a certain frequency response which is proportional to the intensity. This is modeled by passing the log-intensity values through a lowpass filter. **F:** this stage is where the events are generated for each frame, we calculate where is the memorized log-intensity. The number of events that occurred between the frames can be estimated as , note that these is a signed integer quantity where the sign represents the events' polarity. For the next frame we update *.* Note that several events can be generated at once and they will have the same timestamp for lack of better temporal resolution. Several noise mechanisms are simulated at this stage, we will not cover how they are all modeled. The noise that is the most important to us is threshold mismatch, here it is modeled by sampling threshold values for each pixel from a gaussian distribution where (can be also specified by the user). **G:** here the temporal noise caused by shot noise is added. **H+J:** in these stages the output is converted to different formats for displaying and processing.

# 2 Noise in Event Cameras

## ​2.1 Background Activity

Background activity (BA) noise is a name given to a group of noise mechanisms. All these noise mechanisms can cause the generation of events under constant illumination, hence the name background activity noise. Its sources [2] are electrical circuit phenomena such as charge injection, leakage current of the transistors, thermal noise and dark current. Dark current is a current that flows through the photodiode that isn’t proportional to the incident intensity. Since the dark current isn’t proportional to the scene intensity, under bright lighting conditions its effect is relatively less significant [4]. This is because the photocurrent in the diode becomes larger while the dark current does not. Therefore, the effect of this effect on the sensed voltage , as expressed by the following equation is smaller as can be seen in figure 7. The other sources are mainly temperature dependent, and their effect increases as the temperature rises.

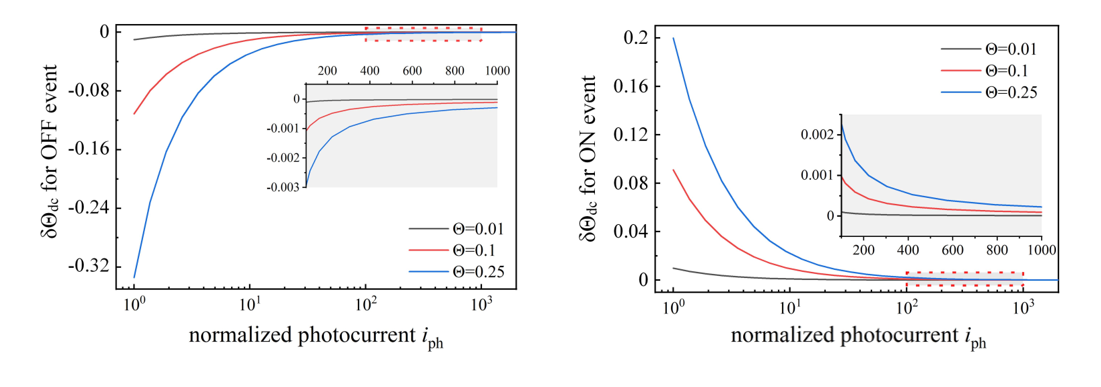


Figure the dark current effect δΘ\_dc as a function of the diode photocurrent, figure taken from [4]

In [2] it was shown that the number of events generated by BA noise can be modeled as a Poisson process, i.e.

It is also said by [2] and [3] that the BA event rate is . BA events have low spatiotemporal correlation with each other because of their low rate and per-pixel behavior. This fact can be used to remove BA events as we will show later. This low correlation can be seen in the BA noise example in figure 9.

## 2.2 Shot Noise

Shot noise is a type of noise caused by the quantum nature of photodiode photon absorption. Each time a photon is absorbed by the photodiode there is a probability that it will cause an electron to generate, and with a probability it will not be generated. This causes the photocurrent to be random and can cause the detector circuit to randomly exceed the pixel's event threshold. It has approximately Poisson statistics and creates roughly the same amount of ON and OFF events [5]. The shot noise events rate is approximately 50 times higher than the BA events rate. It is also dependent on the illumination of each pixel so it can't be filtered simply based on a lack of correlation to the input.

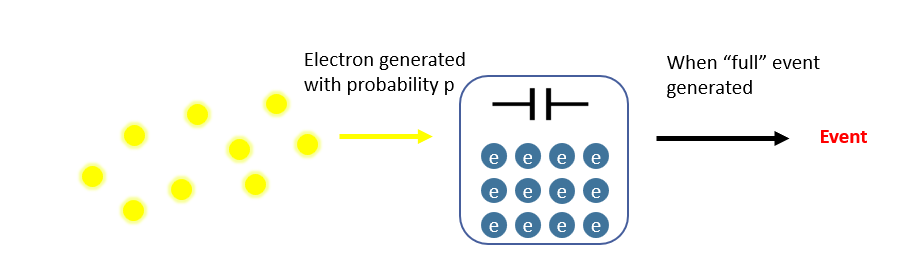


Figure Shot noise effect in event cameras illustrated

## 2.3 Hot / Cold Pixels

DVS sensors often have "hot pixels" which continuously fire events at a very high rate even at the absence of an input, as can be seen in figure 9. This can be caused by abnormally low thresholds or reset switches with very high dark current [3]. On the same principle "cold pixels" are pixels with abnormally high thresholds that will therefore barely generate any events.

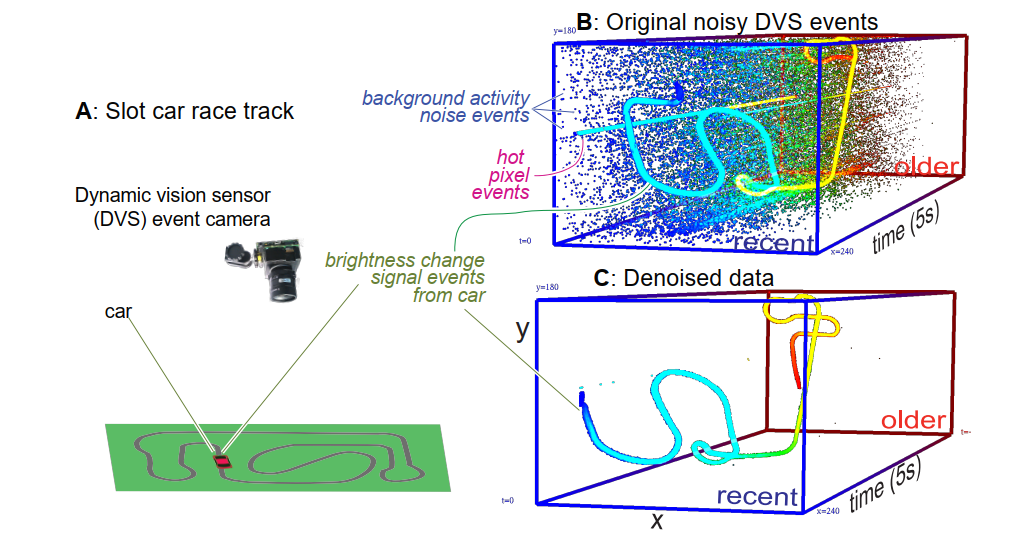


Figure example of event camera output of a slot car racing A: slot car system B: raw data, a hot pixel can be seen as well as background activity noise C: denoised data – only events correlating to the car are left. Figure taken from [5]

Hot pixels can be filtered quite easily, as they are very noticeable as can be seen in figure 9. By excluding highly self-correlated pixels we can remove the hot pixels in the event camera [5].

## 2.4 Threshold Mismatch

Threshold Mismatch means that the contrast threshold is not uniform for all pixels as we would like to set it because of process variation in the manufacturing of the DVS pixels. Process variation means that the transistors and diodes in each pixel are different from each other. The main source of mismatch is the relative mismatch between differencing circuit reset level and comparator thresholds [6]. That is because transistor mismatch is in the order of 30% while capacitor mismatch is only in the order of 1%. Moreover, the comparator does not contain an offset compensation mechanism (a circuit topology that reduces the effect of transistor mismatch) which makes it sensitive to mismatch. [6] measures the effect of threshold mismatch by stimulating the sensor with a black bar with linear gradient edges moved at a constant projected speed. Then the number of events per pixel were counted, the results can be seen in figure 10. This process was repeated for several threshold settings. As can be seen the distributions become wider as the threshold becomes lower which is expected as more events are generated.

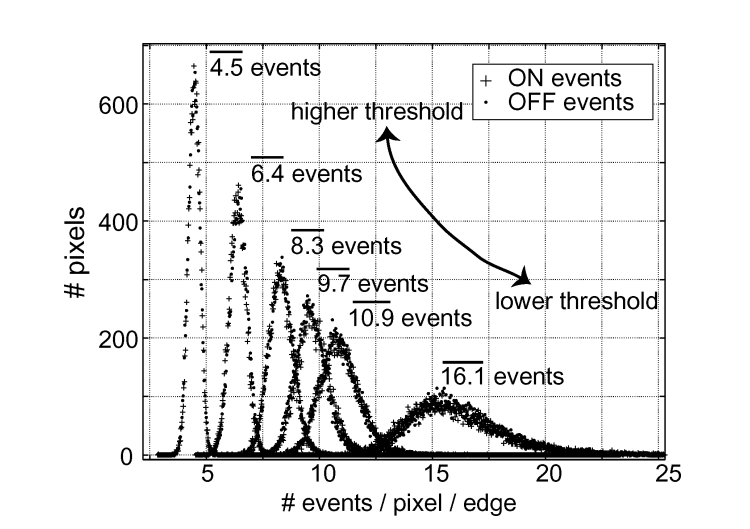


Figure Distributions of the number of events recorded per pass of the bar for 40 repetitions per pixel per edge of the bar, the barred number represents the average number of events per pixel per edge for each threshold setting. Figure taken from [6]

The results seen in figure 10 can be used to calculate the standard deviation of the threshold mismatch from calculating the standard deviation of the number of events , and measuring the contrast change . This can be done with the following relations:

By applying this we get the results is figure 11 and we get a standard deviation of 2-3.5%.

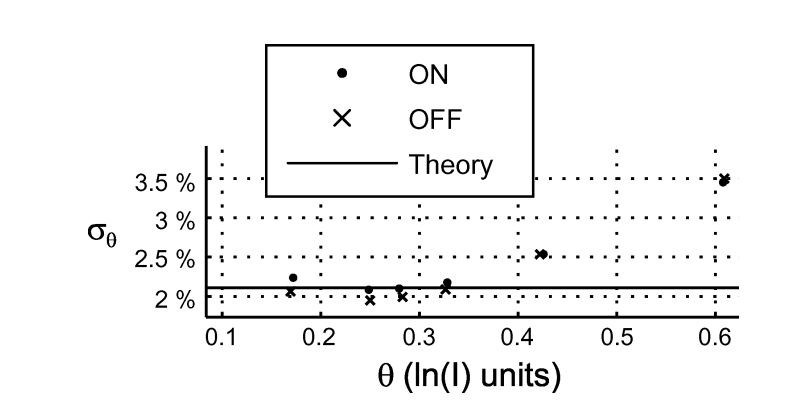


Figure The measured standard deviation of the threshold in % changes of illumination, the theoretic line is predicted mismatch for 10mV relative comparator mismatch. Figure taken from [6]

## 2.5 Threshold Mismatch Event Generation Model

Threshold mismatch effects the event generation and causes each pixel to react differently to the same inputs. Therefore, we would like to update our event generation model to include this effect as well as understand the impact on the sampled event data streams. For each pixel , a sequence of events is triggered by

Where is the difference between the log intensity values of the last two events at pixel . This model includes several new properties. First, it allows each pixel to have its own threshold and by that, it accounts for threshold mismatch. In addition, the model allows the ON and OFF threshold to be different by including a bias parameter , many other works do not allow this in their model, but we chose to include it like in [7] for generality. The model also allows the parameters to be time dependent, meaning they might change while the camera operates. In this work we assume that they are constant in time for each pixel for simplicity. The parameter is called the pixel-wise threshold value and is called the pixel-wise bias value. The effect of the parameter is illustrated in figure 12. In 12(b) for instance, ON events are created because of bigger changes than OFF events as illustrated by bigger jumps in the sampled signal.

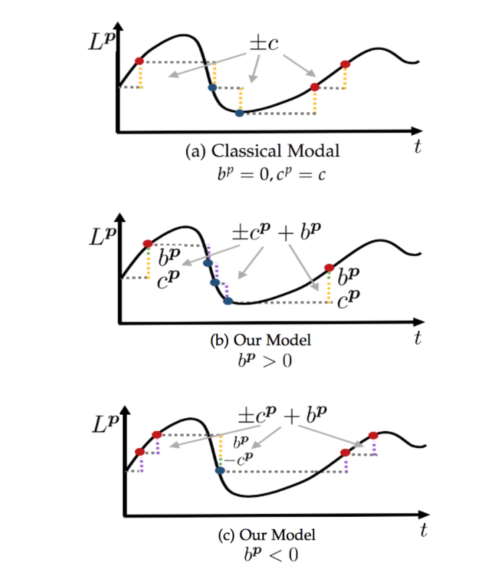
****

Figure Event Generation Model Comparison – (a) the classical model without bias (b) positive bias (c) negative bias. Figure taken from [7]

To further understand how this affects the pixels we can define 4 types of special pixels as seen in figure 13.

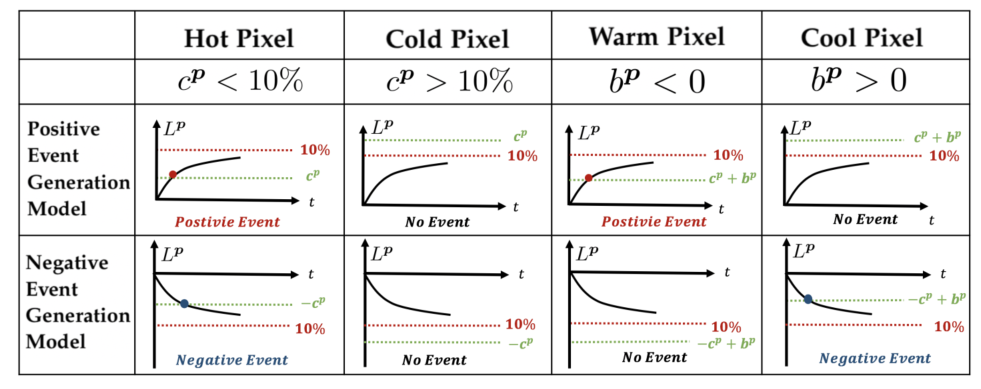


Figure Special pixel types created by threshold mismatch. Figure taken from [7]

# 3. Simulation Environment

## 3.1 Implementing a Background Activity Filter

In order to familiarize ourselves with the data output of an event camera, we examined existing event camera data streams available online. As a first step, we used the simple tool from a library called AedatTools to convert the ".aedat”, which is popular for event camera data, to a data structure usable in a Matlab environment. This allowed us to view the same event data streams which are used as benchmarks for filtering algorithms shown in the literature.

Upon viewing several datasets, we immediately noticed that the most noticeable form of event camera noise is general background activity, indicated by sporadic events occurring in static or empty parts of the scene. This means that before attempting to focus on any specific type of noise (for either detection or filtering) we would need to clear at least a portion of the background activity.

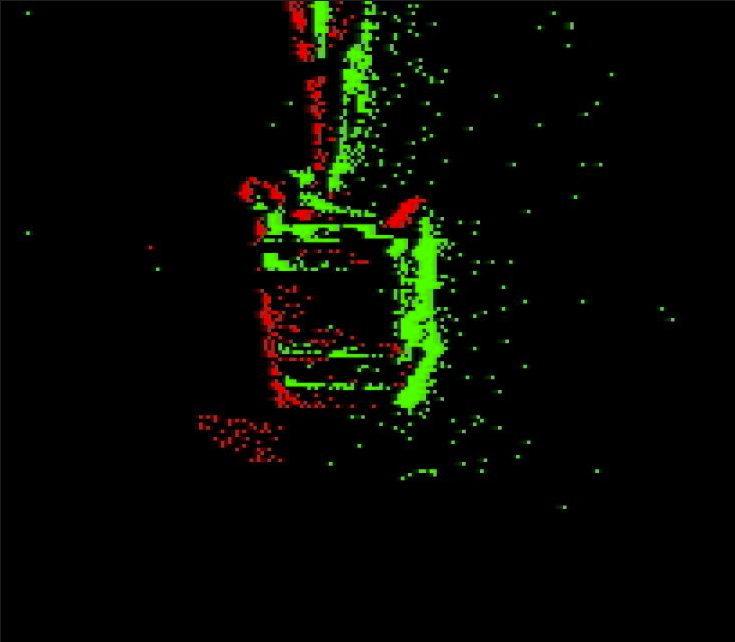


Figure Example background activity in Event Camera Data (box.aedat)

After researching different forms of BA filters, we decided to implement a filter which is based on spatial and temporal correlation of events. These filters utilize the assumption that a noisy event has lower correlation to adjacent events. In other words, a true event is usually characterized by the appearance of other events (of a similar or opposite polarity) in the same neighborhood of pixels around the event’s timestamp. In contrast, a noisy event is sporadic, and is usually singular in its pixel neighborhood.

The above operating principal is described in the following expression ‎2]:

Meaning, an event , characterized by parameters (polarity, coordinates, and timestamp respectively) is not considered as background activity, if there exists some event that satisfies: (1) has a timestamp within a window of event , (2) has coordinates within a distance of pixel from the original event.

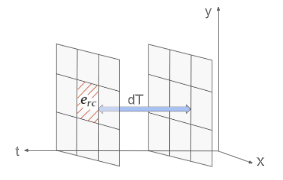


Figure Visual representation of spatial and temporal correlation of an event to test BA. Figure taken from [2]

There are several different implementations of this basic framework, designed to optimize the memory and time complexity of the filtering algorithm, such that the algorithm could operate in real time with minimal computing requirements. For our purposes, we chose a simple post-processing filter which iterates across all events and maintains a memory matrix of the latest events of each pixel. In each iteration, an event is tested. The most recent event saved in the memory matrix, which lies within an 8-neighborhood of the pixel, , is extracted. If the timestamp of is within of the tested event , the event passes spatial correlation and is considered a valid event. Otherwise, the event is discarded, though it can still be used as reference for correlation of future events.

This type of filtering works best for events that are both spatially and temporally isolated. Its main points of failure are (1) Heavy BA could generate two adjacent invalid events, meaning if the look-back window is too large, an invalid event will validate a second invalid event. (2) In a case where a change in the scene is characterized by a steadily growing number of events, the initial ‘catalyst’ events would be erroneously filtered out.

Following are results of the BA filtering algorithm:

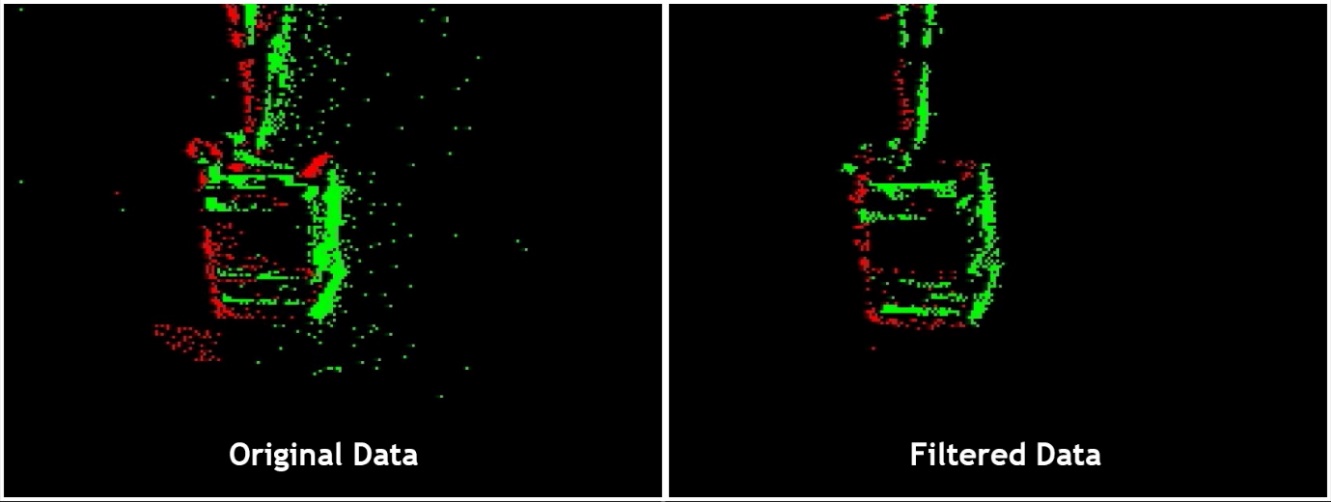


Figure Application of our BA filter on event camera data - box.aedat

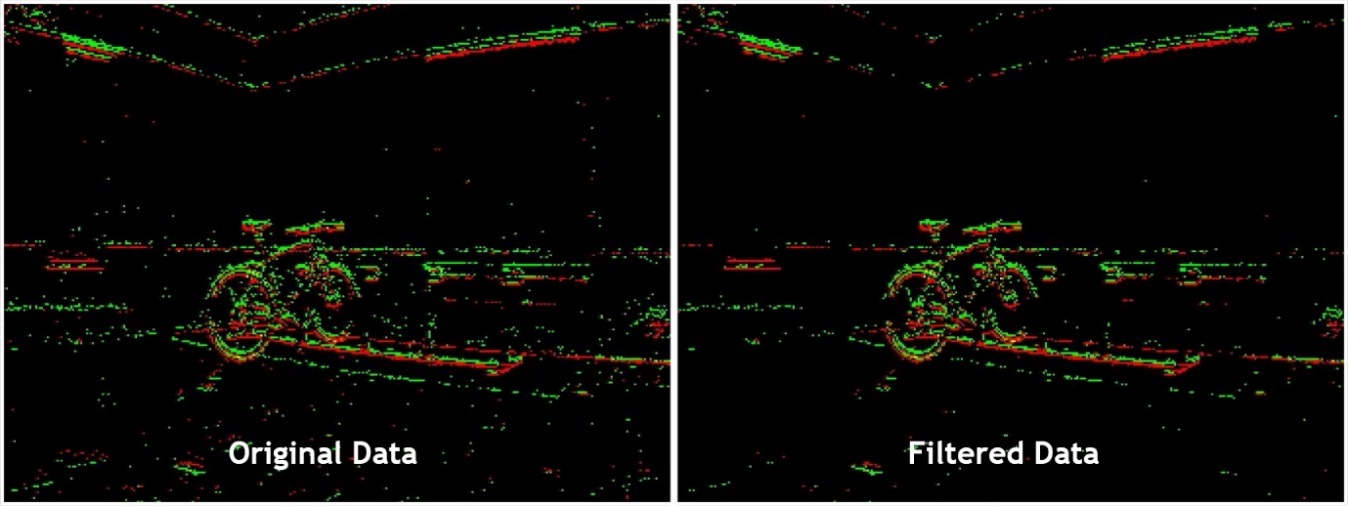


Figure Background activity in Event Camera Data (bike.aedat)

We can see that even when applying a simple BA filtering algorithm, the result is a cleaner frame, with more emphasis on the main components of the scene. However, these components are comprised of fewer events.

## 3.2 Implementing a Video-to-Event Simulation Environment

As we narrowed our focus to threshold mismatch detection and correction, it became necessary to isolate our datasets, such that they include only noise originating from threshold mismatch. However, this is not a simple task, since all types of noise explored in our work and detailed in this document can occur simultaneously, with no clear indication of which type of noise is responsible for a specific invalid event.

This led us to create our own, Matlab-based, video-to-event simulation tool, inspired by the Huang et al [3] V2E tool discussed above. Unlike V2E, this tool would exclusively simulate threshold mismatch noise. The flow of the tool is illustrated in figure 18. Another significant difference is that this tool would not include the Slow-motion interpolation feature of the original V2E, which under performs on synthetically generated videos. The interpolation broke symmetries that were in the video before the interpolation. We wished to use these symmetries to clearly see the threshold mismatch effect in our data. For example, we wanted to generate uniformity in the y axis and the interpolation caused blurring in the image boundaries.

As a first step, the tool loads a video, splits it to frames, and converts each frame from RGB to grayscale format. This is meant to mirror the illumination level measured by event cameras. Secondly, two threshold masks (of negative and positive polarities) are generated. These masks are matrices of a similar size to the original video frame, with values that represent a threshold illumination value that is non-uniform across pixels.

For most of our simulations, we chose to use a normally distributed random function that shares a single expected value and variance for all pixels. A benefit of creating a custom tool is that many different configurations could be created; For example, we could use different normal distribution parameters to distinct regions of the frame. This could aid in simulating a mismatch that is spatially correlated (similar in principle to gross failing areas in chip manufacturing).



Figure Custom V2E tool flow diagram, based on a figure from [3]

The following steps are similar to V2E; the grayscale illumination level of each frame is passed through a lin-log function (discussed above), and subsequent frames are subtracted from each other. If the resulting difference surpasses the corresponding value of either positive or negative polarity threshold matrices, an event is generated. Else, the difference value is saved such that an event can be generated after accumulating sufficient illumination difference. This is meant to mirror the effect of the capacitor and reset mechanism in the event camera circuit.

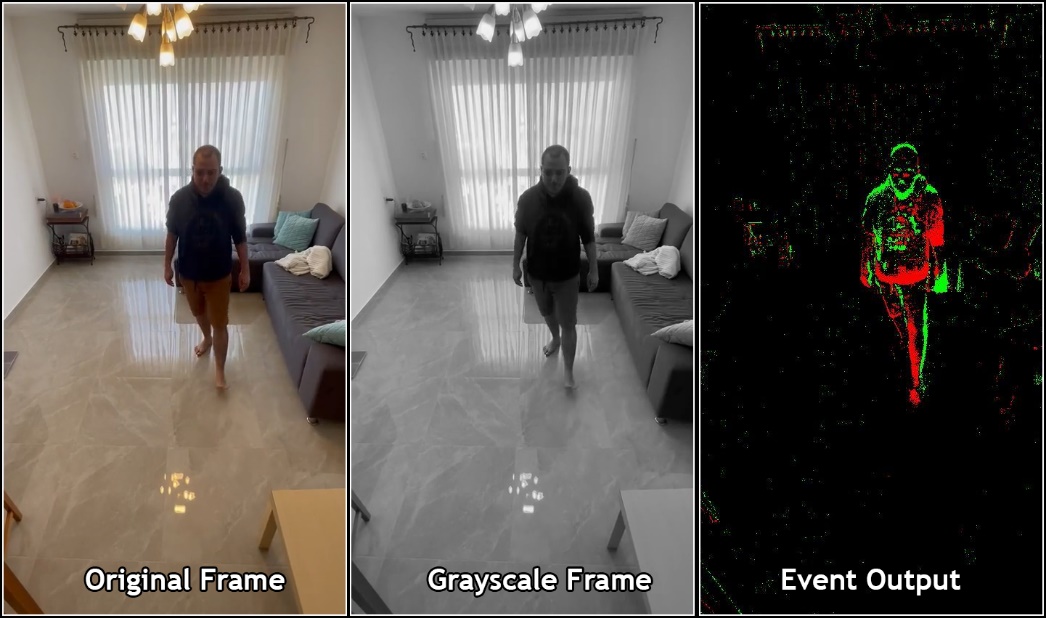


Figure Example frame of the custom V2E tool output

To demonstrate the effect of the added threshold noise, we applied our simulation tool on custom-made videos. Following are some of the results we have achieved with this tool:

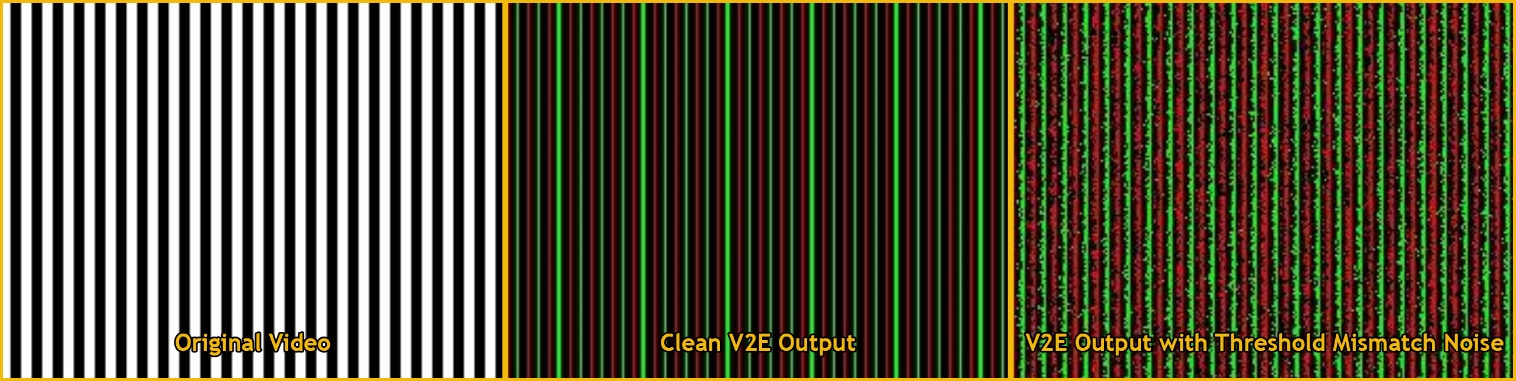


Figure Simulation tool applied on custom moving vertical bars video

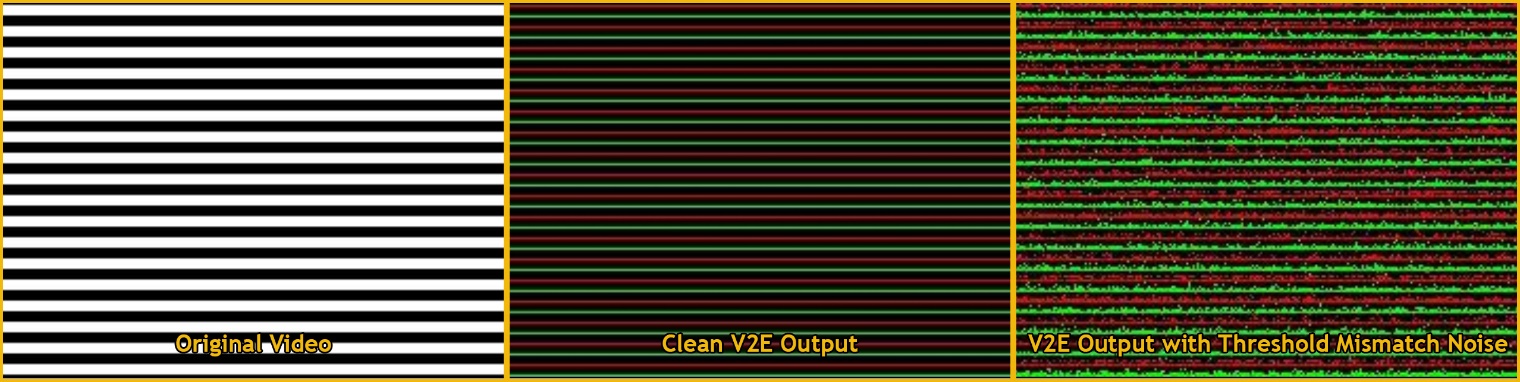


Figure Simulation tool applied on custom moving horizontal bars video



Figure Simulation tool applied on custom uniform gradual flicker video

# 4. Threshold Correction

## 4.1 Threshold Estimation Algorithm

Ziwei Wang et al [7] propose 3 algorithms for contrast threshold estimation in 3 different scenarios. All the algorithms are based on solving a Least Squares (LS) problem, meaning an overdetermined linear system of equations. A general LS problem is of the form where is a matrix and are vectors. The LS solution is the vector that minimizes the expression , where is the norm. This problem has an analytical solution, .

The first two methods assume a DAVIS sensor, meaning that it outputs both a standard event stream as in DVS cameras as well as a synchronous illumination measurement like a traditional VIS sensor. The first method is performed offline i.e., as a prior calibration measurement and is hence called offEI (Offline Event + Intensity). The second method is performed online i.e., during the camera operation and is hence called onEI (online Event + Intensity). This may be required in the case where the threshold is assumed to change during operation as seen in the extended model in section 2.5. And the third method assumes only an event stream output and is performed offline; therefore, it is called offE (Offline Event only). We are interested in the event only case and therefore use the third method for our estimation, and this is the only method that will be explained here.

First, the event stream is divided into temporal segments . The algorithm is based on a relation from equation (5), from which we can conclude that for a time segment the log intensity change follows equation (7).

By rearranging the terms, we can get equation (8).

We want to use this formula to do a linear regression between the polarity sum , and the number of events . To that 2 assumptions are made on the scene used for the estimations. The first assumption is that for each timeframe the illumination is similar between the different timestamps, meaning that the difference is small.

The second assumption is the scene is equally textured, meaning that the variance of the intensity difference is equal between all pixels.

To now estimate the contrast threshold, we do a linear curve fitting for equation (8) for each pixel. We fit to the following model:

Based on the values of the polarity sum and number of events for all the time segments. That is done by solving the following LS problem.

We perform the curve fit for each pixel and calculate the RMSE (Root Mean Squared Error) between the linear fit and the data points. Denote the RMSE for pixel as , and the estimation of the contrast threshold is given by:

where the median is performed between all the pixels. It is important to note that the algorithm does not have a measurement of the actual illumination. Therefore, it only gives a measurement of the contrast threshold relatively to the other pixels. For example, if it means that its contrast threshold is three times bigger than the one who has pixel that has the median values of RMSE. To estimate we can observe equation (8) and get that it relates to like so:

## 4.2 Threshold Correction Algorithm

To approach the issue of threshold correction we want to propose a different viewpoint on the event camera operation from a sampling and reconstruction angle. For a given pixel, the sensed log intensity value can be viewed as a signal that is sampled with a single event camera pixel at an undesired threshold value due to threshold mismatch. We want to propose an algorithm that converts the stream as if it was sampled at the desired threshold . From the sampling perspective this problem is very similar to changing the sample rate of a sampled signal in discrete time. This similarity inspired us to come up with our solution.

The threshold is analogous to the sampling interval, a high threshold will output less events and a coarser sampling resolution and a low threshold will output more events and a finer sampling resolution. The threshold conversion scheme is shown in figure 24 with the sampling analogy.

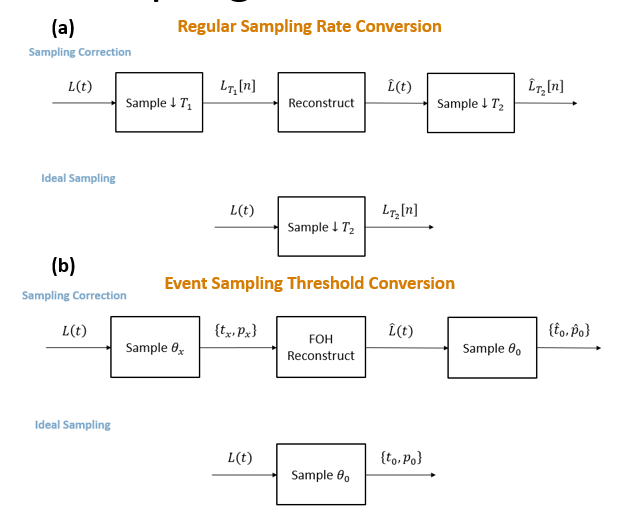


Figure (a) Rate conversion scheme for regular sampling. (b) Threshold conversion scheme for event sampling for both a and b, ideal sampling shows sampling at the wanted rate/threshold initially and the sampling correction shows a resampling in the desired rate

To perform the conversion, first we reconstruct the stream to a continuous time signal , and then we sample the reconstructed signal at the desired threshold . To perform the reconstruction from event stream to a continuous time signal we use a simple First Order Hold (FOH) reconstruction algorithm. FOH means to simply connect each adjacent pair of sampled points with a straight line. Since here the sampling isn't done at a constant interval, the FOH is a little different from the known case for discrete signals. We go over all the timestamps and for every timestamp we know that the amplitude increased by if the polarity is positive and decreased by if the polarity is negative. Figure 25 illustrates the FOH reconstruction for .

Figure 26a shows an example of the FOH on a sinusoidal signal, when the change is fast the reconstruction matches the real signal and when it is slow the reconstruction is inaccurate. That is because the signal is not linear but for fast changes the time interval between events is short and hence the linear approximation of the FOH fits the signal better. Figure 26b show the threshold correction results compared to directly sampling with the desired sample. The peak of the sinusoid is where the correction algorithm misses the ideal sample since the sampling there misses the peak value since it does not cross the threshold.

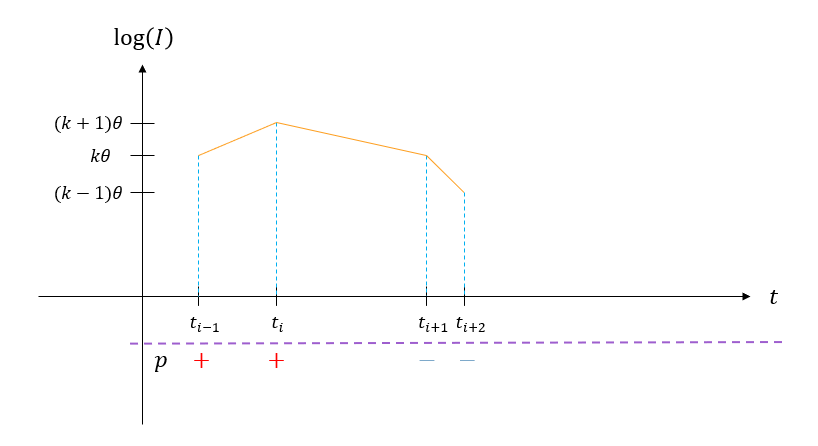


Figure First Order Hold reconstruction illustration of the event sampling, the plus and minus below the purple line mark the polarity of each event.

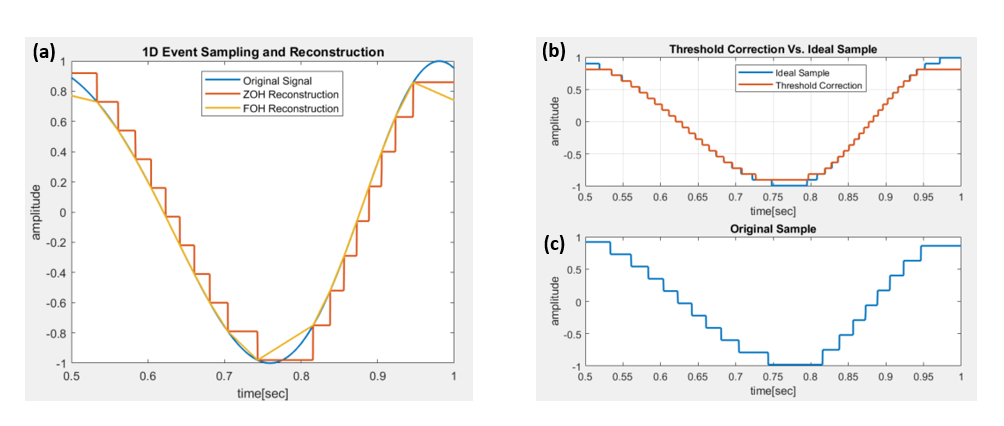


Figure (a) FOH reconstruction illustration – the original signal shows the log intensity value, ZOH reconstruction is a stepwise signal that changes by the threshold value each time an event is recorded, FOH is the reconstruction as explained in the text. (b) shows the results of the ideal sample straight with the desired threshold (blue) and the threshold correction result (orange) (c) the Original sampled signal ZOH reconstruction

The reason we choose to use FOH reconstruction is because of its simplicity. This simplicity can be used to perform the conversion efficiently without needing to calculate the reconstructed signal. The corrected timestamp can be computed analytically from the linear fit model. To do that keep a variable accumulating the log intensity change from one event to the next. Because we can compute the log change in intensity of the reconstructed signal analytically from the following formula:

Where is the remaining log change from the previous time frames, and is the threshold of the event according to the polarity. From equation 15 we can calculate the exact time that the reconstructed signal crossed the desired threshold and can therefore compute the correction in real time when using the camera. This is a major advantage of the algorithm because it means it can be implemented on the sensor itself and doesn’t require external post processing.

## 4.3 Incorporation in General Filtering Scheme

To practically apply the correction algorithm when processing data we want to explain the denoising pipeline and where threshold mismatch correction fits inside of it to provide the best performance. Since the background activity filters aren’t based on pixel similarities in number of events or fine levels of timestamp mismatch, the threshold correction algorithm will not improve the denoising performance and hence can be applied before or after the background activity filter. On the other hand, threshold correction and estimation highly benefit from BA noise reduction. That is because the BA events aren’t related to the actual intensity values that the estimation and correction are based on. Therefore, the BA noise filtering should be performed before the threshold estimation offline, as well as applying the filter before the threshold correction algorithm.

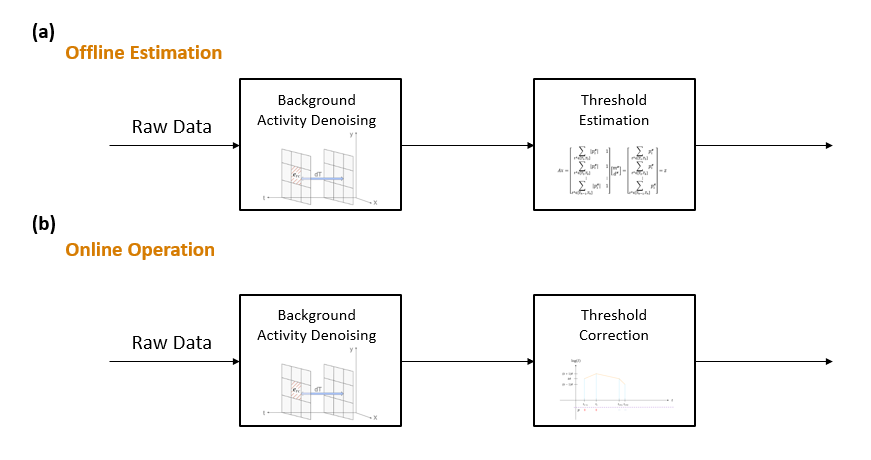


Figure (a) the offline threshold estimation scheme (b) the online threshold correction algorithm operation

# 5. Performance Analysis

## 5.1 Threshold Estimation Performance

To test the performance of the threshold estimation algorithm we generate synthetic event data with the v2e tool introduced in section 3.2. We generated two scene types to fit the assumptions of the estimation algorithm. The first one we named as the flicker scene, where all pixels get a linearly changing intensity value increasing to the maximum intensity value and decreasing to the minimum intensity value. These changes happen periodically through the time of recording. The second scene we named the moving gradient scene. It is a linearly rising and decreasing intensity scene in the x direction, and this pattern moves to the right in the x direction as can be seen in figure 28a.

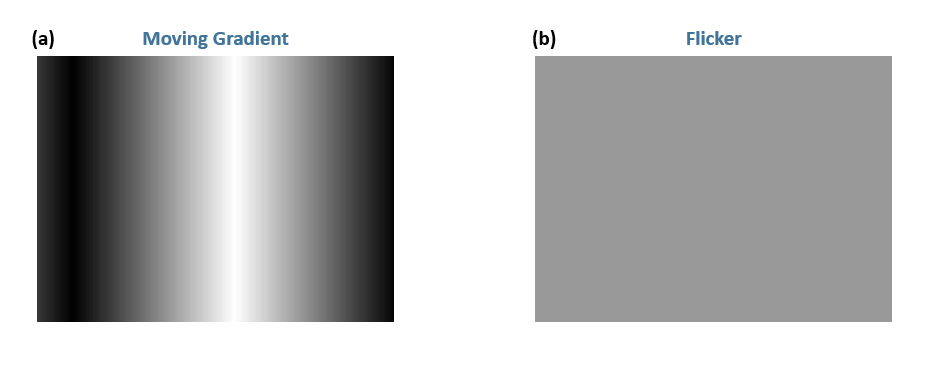


Figure (a) Moving Gradient scene snapshots (b) Flicker scene snapshot - all pixels change uniformly

We generate event streams from these scenes using our v2e tool and then apply the threshold estimation algorithm on the resulting streams. Using the known thresholds of each pixel we can compare the results to the estimated results and see how the algorithm performs. Ziwei Wang et al [7] tested the algorithm only on real event data. The performance of the estimation was measured via the effect of the known thresholds on the quality of video reconstruction from the event data. Therefore, we think that this experiment can improve the understanding of the algorithm. The results are shown in Table 1. The error calculated as the RMSE . The estimation error is 5-7%

Table Estimation Algorithm Results – the average absolute estimation

|  |  |  |
| --- | --- | --- |
|  | Flicker Scene | Moving Gradient Scene |
| Positive Thresholds Error | 7.07% | 6.04% |
| Negative Thresholds Error | 5.14% | 6.38% |

Figure 29 shows a color map of the estimation error for both scenes. The map in figure 27b shows a line pattern similar to the original scene. The flicker scene shows an estimation error without a scene related pattern that emphasizes the importance of estimation scene selection to fit the algorithms' assumptions.

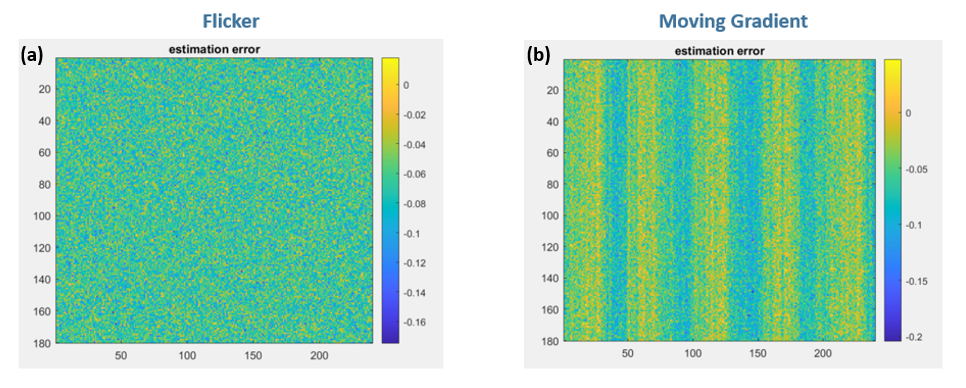


Figure The estimation error of the threshold values. (a) estimated thresholds and (b) actual thresholds in the flicker scene. (c) estimated thresholds and (d) actual thresholds in the moving gradient scene.

## 5.2 Threshold Correction Performance with Perfect Estimation

To test the performance of the correction algorithm first we assume that the algorithm perfectly knows the thresholds of each pixel. Natural scenes are characterized by sudden changes in illumination hence, to test the performance we use a step like stimulus as can be seen in figure 30. To mimic the actual illumination that the circuit will sense we pass the signal through a single pole low pass filter to mimic the frequency response of the sensing circuit. To better fit these step signals we slightly change the FOH reconstruction such that if the time interval between 2 events is greater than some value than the reconstruction is stepwise rather than linear. That is because the linearization in the constant part of the signal does not fit its nature.

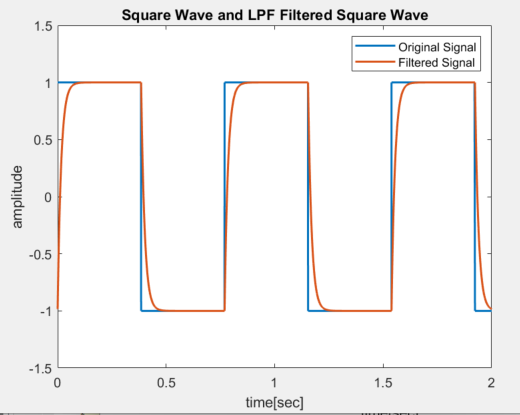


Figure The Stimulus for testing the threshold correction algorithm the blue line is the step signal and the orange line is the result of passing it through a single pole low pass filter to mimic the frequency response of the sensing circuit

100 threshold values are randomly generated, representing 100 pixels with threshold mismatch. Then the same signal shown in figure 30 is sampled by each pixel with its different threshold value. The result of the sampling is shown in figure 31b. we can see non uniform responses for the different pixels. Then we apply the correction using the exact thresholds of the pixels to reconstruct the signals. The result of the correction algorithm is shown in figure 31a. After the correction the response is much more visually uniform, the timestamps are more even, and the steps are of equal sizes.

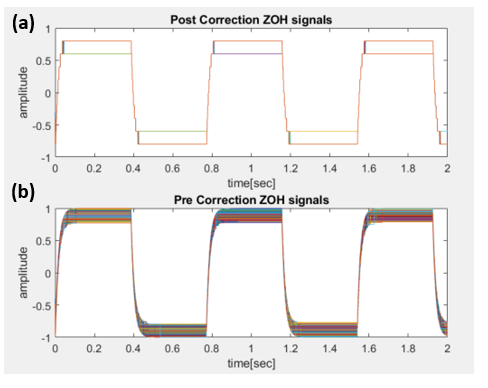


Figure The results of applying the correction algorithm. 100 equal signals were sampled with threshold mismatch and then with knowledge of the exact thresholds of each pixel the threshold correction is applied (a) all the signals after the threshold correction displayed as steps when (b) all the originally sampled signals with threshold mismatch

## 5.3 Threshold Correction Performance with Estimation Error

As we saw the estimation algorithm isn’t perfect and has an estimation error. To test the impact of the estimation error on the correction algorithm we repeat the experiment from the last section. The difference is that this time we do not use the exact thresholds but rather add a random gaussian estimation error to the actual thresholds and use them for the reconstruction part of the correction algorithm.

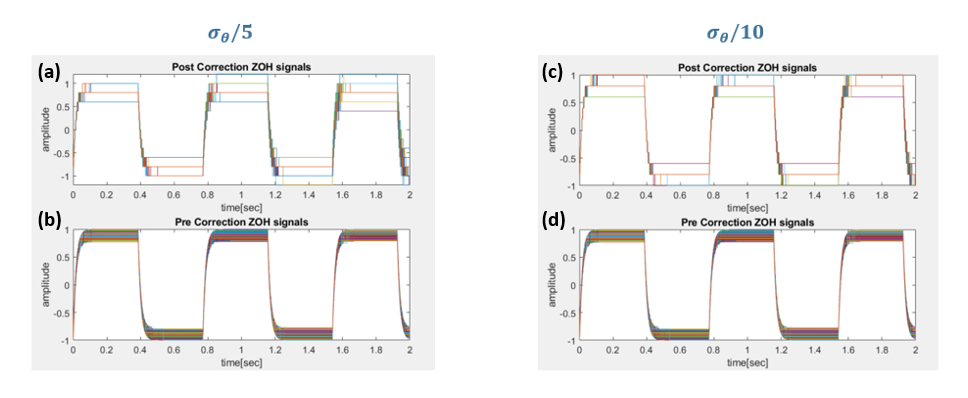


Figure The results of applying the correction algorithm with estimation error, (a) all the signals after the threshold correction displayed as steps when (b) all the originally sampled signals with threshold mismatch

Figure 32 shows the result of applying the correction algorithm with estimation error. We can notice in figure 32a that as the time progresses the reconstructions are less and less uniform. That is because the estimation error accumulates as the scene progresses since each event adds a small estimation error to the reconstruction. The accumulating error is very problematic since even if the estimation error is small it is accumulated and therefore becomes significant as the recording time increases. Figure 32b shows the same effects but they are less significant since the estimation error is smaller.

Previous work such as [1] refer to the first and last events at each change as the most important. Therefore we also wish to use that as a metric for the algorithm performance. The correction algorithm should create uniformity between the pixels, so we use the standard deviation of the first and last timestamps in the change to test their uniformity and the performance of the algorithm.

Table Uniformity analysis with and without estimation error – the table shows the standard deviation of the first and last timestamps of one of the step changes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Pre-Correction | Post-Correction 0 Estimation Error | Post-Correction Estimation Error | Post-Correction Estimation Error |
| Std of first timestamp of the edge | 0.57ms | 0.46ms | 0.74ms | 0.72ms |
| Std of last timestamp of the edge | 15ms | 2.23ms | 12.2ms | 12.3ms |

The first timestamp variation is small in the original data since when a fast change occurs it crosses all thresholds at close timestamps. Because of that there isn’t a lot of room for improvement in uniformity as we can see that even with the ideal estimation the improvement isn’t significant. With the estimation error the algorithm degrades the uniformity. On the last timestamp variation the ideal algorithm improves the pre-correction results from 15ms to 2.23ms, almost an order of magnitude improvement. Even the algorithm with the estimation error reduces the variation to 12.2ms, which shows for the uniformity improvement.

# 6. Conclusion

In this work we introduced an algorithmic approach to the threshold mismatch effect in event cameras. Applied a scheme of estimating the threshold values offline before the camera operation and tested 2 types of scenes. With the estimated thresholds we propose a novel correction algorithm based on sampling theory.

The estimation algorithm achieved 5-7% RMS estimation error for both the positive and negative threshold values as well as for both scenes. Contrary to the original paper [7] that suggested the algorithm we tested its performance in simulation form. We adapted a v2e [3] like tool to create event streams from synthetic high-resolution videos we created in MATLAB. Then we applied the estimation algorithm and compared the estimated threshold values to the actual ones. The original paper [7] tested the algorithm's effect on video reconstruction, our experiment compares the estimated results to ground truth.

The correction algorithm provides a different approach to the correction and denoising problem. Most methods refer to classifying the events as noise or signal events and remove the noise events. We take an approach of "translating" the data from one threshold value to another rather than working on single events and searching for correlations. The algorithms' results can be seen in figure 31. Visually, the uniformity is improved after the correction algorithm both in the number of events and the timestamps. We also analyzed the first and last timestamps' uniformity as shown in table 2. Testing the standard deviation of these timestamps between the different pixels. This is a new metric for the severity of threshold mismatch. The metric is simple to apply and understand and does not require ground truth to compare the results to. The first timestamps are relatively uniform even before the correction since the change is fast and therefore all the pixels sense it at close times. The last timestamp represents a slower change and therefore the linearization fits it better.

# 7. Further Work

This work is only the first step in exploring the noise model of threshold mismatch and the means of correcting it. There are many options to expand on this work in both threshold estimation and threshold correction.

For threshold estimation, different scenes can be explored beyond the two we covered (flicker and moving gradient), as the optimal scene could yield better estimation results. Furthermore, this scene could be replicated in a lab to attempt estimation on actual event camera pixels, though this would require dealing with additional types of noise. Furthermore, this work assumed the DVS in question cannot measure the illumination level. However, many event cameras offer this option using an additional APS such as the DAVIS cameras. Adding the measured illumination data to the estimation algorithm, in the form of additional input vectors in the linear regression problem, could potentially improve the estimation accuracy (this is also explored in the article by Ziwei Wang et al [7], OffEI method). Finally, it could also be useful to alter the estimation algorithm such that it could run online (estimation changes dynamically while shooting a scene). This is because some sources suggest that the pixel threshold is not time invariant, and is scene-dependent, despite the assumptions made in this work. Online calibration also makes the sensor easier and cheaper to use since it doesn’t require additional lab equipment to operate.

In threshold correction, developing better performance metrics could be a worthwhile improvement to better judge the effectiveness of the algorithm. We used pixel-uniformity-based metrics to test the performance of the algorithm working on the timestamp's uniformity. This fails to consider the number of events for example and isn’t translated directly to performance improvement of the sensor yet. Testing how the algorithm effects computer vision algorithms on corrected event streams may also be interesting as a performance metric. Additionally, as shown above, the threshold correction algorithm is not noise-robust enough in its current state. Introducing even small threshold estimation errors could cause the algorithm to hinder rather than help pixel uniformity. This is also true when considering other types of noise, such as background activity, which were excluded from this portion of the project. Finally, the idea at the heart of this algorithm, which is viewing the threshold correction problem from a signal processing and sampling theory lens, could be expanded upon with alternative methods, such as higher orders of reconstruction instead of FOH or reconstruction methods that better fit signals related to natural scenes, for example step-like signals like the ones we used in the performance analysis.

We are confident that exploring even some of the options suggested above could allow for significant strides in exploring and solving the threshold mismatch problem.

# References

1. Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davidson, Jörg Conradt, Kostas Daniidis and David Scaramuzza, "Event-Based Vision: A Survey", IEEE transactions on pattern analysis and machine intelligence 44.1 (2020): 154-180
2. Alireza Khodamoradi and Ryan Kastner, "O(N)-Space Spatiotemporal Filter for Reducing Noise in Neuromorphic Vision Sensors", IEEE Transactions on Emerging Topics in Computing 9.1 (2018): 15-23
3. Hu Yuhuang, Shih-Chii Liu, and Tobi Delbruck, "v2e: From video frames to realistic DVS events." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2021)‏
4. Jiangtao Xu, Jiawei Zou, Zhiyuan Gao and Jianguo Ma, "Analysis of Input-Dependent Noise in Self-Timed Reset Dynamic Vision Sensor and Its Impact on Data Quality", IEEE Sensors Journal (2019)
5. Shasha Guo and Tobi Delbruck, "Low Cost and Latency Event Camera Background Activity Denoising", IEEE Transactions on Pattern Analysis and Machine Intelligence (2022)
6. Patrick Lichtsteiner, Christoph Posch and Tobi Delbruck, "A 128 x 128 dB 15 Latency Asynchronous Temporal Contrast Vision Sensor", IEEE Journal of Solid-State Circuits (2008)
7. Ziwei Wang, Yonhon Ng, Pieter van Goor and Robert Mahony, "Event Camera Calibration of Per-pixel Biased Contrast Threshold", Australasian Conference on Robotics and Automation, ACRA (2019).
8. C. Scheerlinck, H. Rebecq, T. Stoffregen, N. Barnes, R. Mahony, and D. Scaramuzza, “Ced: Color event camera dataset,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2019, pp. 0–0.
9. E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, “The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam,” The International Journal of Robotics Research, vol. 36, no. 2, pp. 142–149, 2017
10. Samuel K. Moore, ”Prophesee’s Event-Based Camera Reaches High Resolution” https://spectrum.ieee.org/prophesees-eventbased-camera-reaches-high-resolution

A Git repository containing a lot of useful information and resources about event cameras: Event-based Vision Resources - <https://github.com/uzh-rpg/event-based_vision_resources/blob/master/README.md#event-denoising-1>