Event-based Stereo Vision
Practical Course Computational Neuroengineering

Seth Siriya, Alexander Matthies, Nico Hertel

August 2, 2018

Abstract

Many computer vision tasks rely on a fast and precise depth estimation, e.g. when planning motion. In this report, a novel method for stereo vision is described that uses an event-based-vision-sensor instead of the common frame-based-cameras to create a sparse and low-latency data stream. Due to the unique format of the data, usual stereo vision algorithms are not applicable. We therefore constructed a processing pipeline to provide an interface between our local stereo camera setup and a pre-existing implementation of an event-based stereo depth estimation algorithm that is applicable to this unique data stream. We also try to increase the performance of said algorithm by modifying the algorithms in MATLAB and also implementing in C++. Our modification caused the algorithm to linearly scale with matched events, and the C++ implementation was six times faster than the MATLAB-implementation while reproducing the same estimated disparities.

1 Introduction

One common task in computer vision is the estimation of depth for certain objects whether it is for improved object recognition, 3D-navigation of autonomous robots or position tracking for entertainment systems. There exists a variety of methods to estimate the depth of a scene, one of which is stereo vision, where two cameras record the same scene but from a slightly shifted position. The resulting disparity in the captured images can be used to estimate the depth of the objects captured. With increasing demand for precise and fast depth estimation, more and more commercial devices entered the competition, some of the latest being the Intel RealSense, the Sereolabs ZED or the DUO MLX. Those solutions use so called frame-based cameras for their calculations. Here, the camera takes a picture (frame) in a constant interval. With a framerate of 30 FPS (Frames per Second), the whole scene is therefore captured once every $\frac{1}{30}1/\text{s}$ and can be compared by a stereo vision algorithm.

In recent years, thanks to an increase in computational power and large publicly available datasets, those algorithms and cameras improved noticeably. Still, they struggle under non-ideal circumstances such as rapid movement, insufficient lighting or limited computational power. Of special concern is the latency, how long it takes the algorithm to estimate the depth. The smaller the latency, the faster the application can react to the new information. For commercial frame based cameras, this is in the range of a few frames, or 20 ms to 100 ms. Both latency and the accuracy of the estimated depth can be increased with a higher framerate or more computational power, but this usually comes at the cost of a higher power consumption, larger dimensions or a heavier product [Mustafah et al., 2012, Lucas and Kanade, 1981].

In contrast to such frame-based solutions, this paper follows a event-based concept, using a event-based vision sensor. Here, instead of one picture every timestep, single events are recorded with a corresponding time step, X- and Y-coordinates and the polarity of the event. An event here describes the change of illumination of a pixel. This comes with multiple benefits, the most obvious one being the sparse datastream. Instead of recording every pixel at every frame, only the changed polarity is recorded. Other benefits include a very large dynamic range of up to 120 dB, low power consumption of between 5 mW and 15 mW and a latency of about 100 µs to 1 ms [Lichtsteiner et al., 2008]. But due to the difference in the recorded data, conventional stereo vision algorithms (as well as other computer vision algorithms) for frame-based cameras are not well suited to be used with event-based sensors. Therefore, new stereo vision algorithms have to be designed.

In this project, we implemented and tested a new event-based stereo depth estimation algorithm that uses belief propagation proposed by [Xie et al., 2017]. This algorithm uses a passive method for estimation so the sensor is only capturing the ambient light of the scene. In contrast, active methods such as used by the Microsoft Kinect use a infrared light-source to scan the environment. Those methods achieve good stability in real-time operation, but suffer from the rather short range of the IR-beams and possible interference from ambient IR light. Passive methods such as the one we used do not rely on an external light-source and therefore provide a higher detection range and better depth-resolution, but require more computational power and often require visual features for matching the two different outputs from the two cameras.

For this paper, we used the embedded Dynamic Vision Sensor (eDVS) proposed in this paper [Conradt et al., 2009], where already there is already some literature on event matching or stereo vision [Lichtsteiner et al., 2008, Kogler et al., 2011]. The latter tried to match events between the two cameras based on the correlation of difference in time and polarity. While this showed some reasonable good initial results, using only those two characteristics for event matching will lead to some errors due to inconsistency in latency for the events, so called jitter. [Rogister et al., 2012] proposed an extension of said algorithm which also includes geometry constrains and event ordering. This lead to an increase of reconstruction accuracy, but still struggles with resolving ambiguities, where one event can correspond to multiple events on the other sensor. Those algorithms and similar works like [CamuÃ±as-Mesa et al., 2014, Lagorce et al., 2017] are still not able to have a high estimation rate (number of depth estimates in relation to input events) and produce reliable stereo matches. In contrast to state-of-the-art algorithms in frame-based depth estimation, those methods do not account for 3D constraint between neighboring events in the physical world. [Besse et al., 2012] show that both disparity uniqueness and continuity constraints can be used in event-based stereo vision.

Another solution is proposed in [Firouzi and Conradt, 2016], where a cooperative neural network is able to use the temporal information of the data stream for event matching. This allowed for a remarkable high estimation rate compared to other algorithms, but requires certain parameters of the captured scene that cannot easily be predicted beforehand.

Our goal was to become become acquainted with the software and hardware for dynamic vision sensors, and ensure that the event-based stereo depth estimation using belief propagation algorithm [Xie et al., 2017] can easily be applied to new event recordings. Later on, we attempted to improve the performance of this particular algorithm.

2 Materials and Methods

2.1 Hardware setup

The specific sensors used for recording were the DAVIS240C dynamic vision sensors [Brandli et al., 2014]. These dynamic vision sensors (DVS) are capable of asynchronously recording dynamic events, as well as also capturing image frames allowing for comparison between the two. As the goal is to apply event-based stereo depth estimation algorithms to the recordings, two sensors are required to obtain two event streams simultaneously. This was achieved by synchronizing timestamps between the sensors using an audio cable, then connecting the sensors to a computer via separate USB cables. The specific setup is illustrated in Figure 1, which shows the 3D printed mounting holding the setup together, as well as an actuator and tripod on top. The baseline of the cameras was 10 cm.

After connecting the sensors to the computer, the jAER software provided by the Sensors research group at the Institute of Neuroinformatics in Zürich was used to log event data from both sensors simultaneously (http://www.jaerproject.org). The information recorded for each event was encoded in address-event representation, containing time stamp ts, horizontal pixel x, vertical pixel y, and polarity p [Brandli et al., 2014]. Furthermore, active-pixel sensor (APS) information was also available in events, which allowed acquisition of image frames for comparison. The event

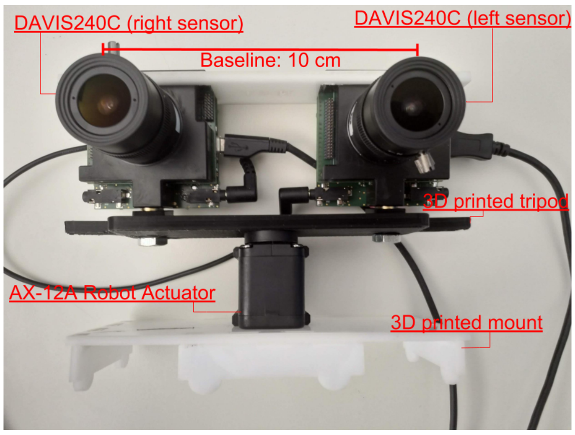


Figure 1: Camera setup

streams containing this information are exported as separate AEDAT 2.0 files corresponding to each camera.

2.2 Data processing pipeline

2.2.1 Construction

As we found that an implementation of the event-based stereo matching algorithm is available online, the goal became to ensure that the local setup could easily interface with the algorithm to obtain disparity estimations from [Xie et al., 2017]. This implementation of the algorithm expects a single matrix of events as inputs, with rows corresponding to events and columns corresponding to ts, x, y, p, camera l and ground truth depth gt_d. The output from the algorithm is a structure containing ts, x, y, gt_d as well as the estimated disparity d. However, the events from the sensors are exported in the AEDAT file format, which is not immediately understood by the algorithm. Therefore the AEDAT file needed to be processed into an appropriate matrix to be understood by the algorithm. The overall pipeline about to be explained is diagrammatically represented in Figure 2.

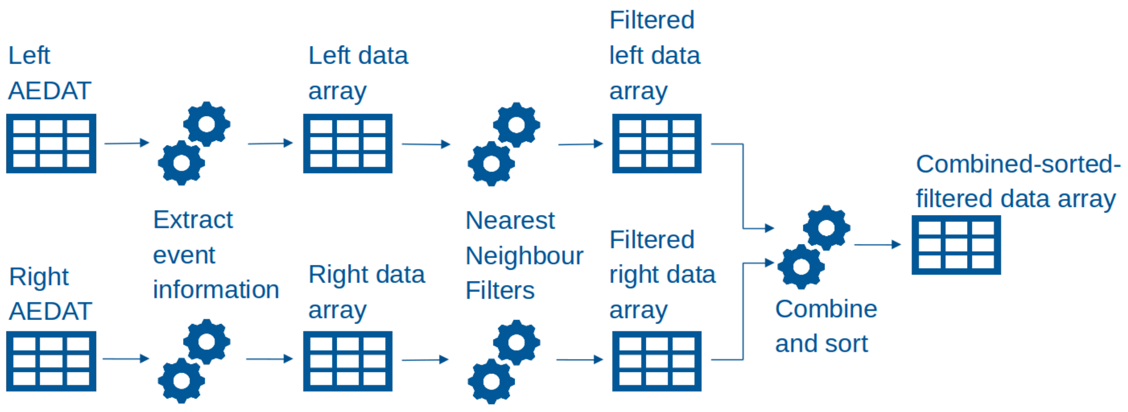

Left AEDAT
Left data array
Filtered left data array
Extract event information
Nearest Neighbour Filters
Combined-sorted-filtered data array
Right AEDAT
Right data array
Filtered right data array
Combine and sort

Figure 2: Processing pipeline from AEDAT to matrix for the event-based stereo depth estimation algorithm.

The first step towards establishing an interface between the sensor and the algorithm was to load all events from the AEDAT files into arrays in MATLAB. From the files, y, x, p and ts could be extracted, but gt_d and l were unavailable. The purpose of gt_d is to allow performance of estimations to be evaluated [Xie et al., 2017]. Rather than being concerned with evaluation, our goal was to establish a pipeline to ensure disparity estimations could be obtained for locally recorded events, and therefore it was decided as fine to set arbitrary values for gt_d in the algorithm.

Concerning l, we note that there are separate event streams corresponding to each camera, and therefore l can be set by keeping track of the streams when combining them later down the pipeline.

The next step was to remove the noise from the individual event streams. Noise in the event streams arise in the form of false alarms, i.e. events detected which should not have occurred. This was addressed through the application of a nearest neighbor filter, which works by removing an event from the stream if no other events occurred in the 8 neighboring pixels within the preceding 30 ms from the same sensor [Czech and Orchard, 2016].

Two separate streams of filtered events corresponding to each camera are now available, but the event-based stereo depth estimation algorithm expects a single a matrix of events. The streams could simply be merged together to achieve this, making sure to set l to indicate which camera the event belongs to, as well as sorting the events by increasing order of ts. Following this step, a matrix containing events from both cameras was ready to be passed to the algorithm.

We should mention that in the original method for preprocessing [Xie et al., 2017], stereo rectification was an important step performed. This was ignored in this project as the main priority was to provide an interface to process data for input into the disparity estimation algorithm. However, the pipeline can be modified to include rectification after the filter and before combining the streams together.

2.2.2 Verification

Following completion of the data processing pipeline, we needed to verify it to be operational. A stereo recording from the DAVIS240C sensor (with AEDAT file extension) was used for this. The specific recording was of the background of a room as the sensor rotates anticlockwise, and was aptly named 'rotate'. This recording was fed into the data processing pipeline. If the data processing pipeline was successfully implemented, then an output event stream with disparities should be generated, and able to be visualized by iteratively plotting these events according to their position and coloring them according to disparity level (or depth).

2.3 Algorithm implementation

The data processing pipeline detailed allows for the depth estimation algorithm to be easily applied to newly recorded data without manual data processing by the user. Following this, attempts were also made to speed up the algorithm, which would allow for the faster acquisition of disparity estimations and improve convenience. These attempts came through the modification of the existing algorithm implemented in MATLAB, as well as a reimplementation of the algorithm in C++.

2.3.1 MATLAB modified

The most important change to the MATLAB code is the way the `stereo_TD` variable is generated, which contains the stereo-matched events and is the output from the algorithm. In the original version it is procedural generated by copying the existing variable and adding the new value (e.g. `stereo_TD.x = [stereo_TD.x; current_x];`). This slows the process down since each new processed event requires all previous events data to be saved again. In the modified version the `stereo_TD` variable is initialized to be an array of the size of unmatched events. It then inserts the data of a matched event at the position of the current event (e.g. `stereo_TD.x(event_index) = current_x;`). After all unmatched events have been processed all empty entries of the `stereo_TD` variable are removed, leaving the same variable as the original version, but reducing the required run time, since the data is not saved unnecessarily often.

2.3.2 C++

Following modification of the algorithm in MATLAB, we believed that performance could be further improved by adapting the algorithm into a compiled language. C++ was chosen due to its popularity as a compiled language, as well as the presence of libraries which implement functions that mimic MATLAB. The biggest challenge faced was the efficient implementation of matrices and multidimensional arrays, as the belief propagation algorithm relies on multidimensional operations. Although this could be implemented via loops, doing so would not be optimizing the code to utilize the multiprocessing capabilities of a computer. The Armadillo library was used to handle this, which is a C++ library acting as a wrapper for various linear algebra libraries, and could be used to implement many of the linear algebra capabilities found in MATLAB (http://arma.sourceforge.net/). The correctness of the implementation was verified by comparing outputs between the MATLAB and C++ implementations for the same input data. After some debugging, the outputs were eventually consistent and so the implementation in C++ was successfully completed.

2.3.3 Performance

Performance of the implementations were tested by timing the duration of the implemented algorithms on various event streams. Three different event streams were evaluated, which were 'box1', 'walking2' and 'rotate'. The first two sets of data were taken from the GitHub repository for the paper on the stereo-matching algorithm [Xie et al., 2017]. As mentioned before, 'rotate' was obtained by processing a recording on the data processing pipeline. Alongside comparing performance, these three streams were chosen as they contain differing number of events and thus the robustness of the algorithms to scaling could be explored. The number of events for the streams are shown in Table 1.

Table 1: Number of events in streams before (unmatched) and after (stereo-matched) applying stereo matching

Event stream	Unmatched events	Stereo-matched events
box1	97179	30536
walking2	636440	249515
rotate	30000001	1110821

3 Results

3.1 Data processing pipeline

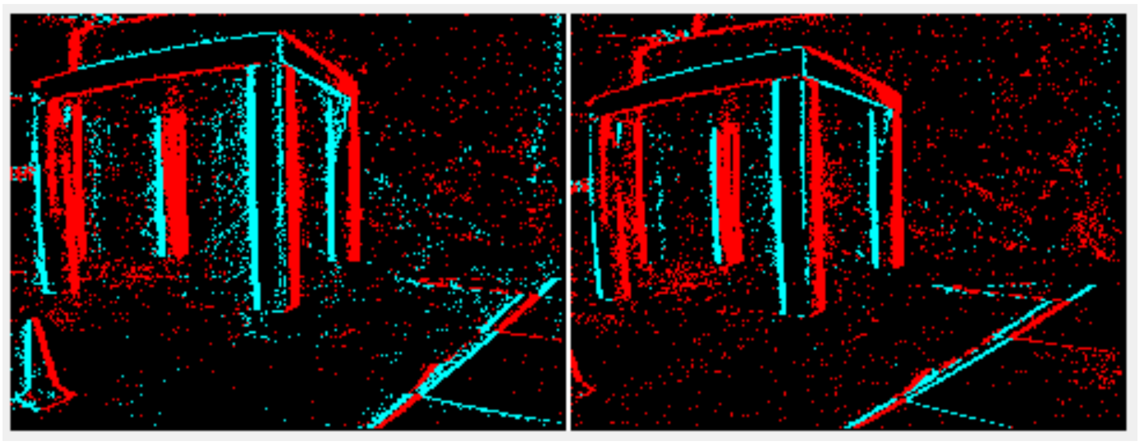
After inputting 'rotate' into the data processing pipeline, an output event stream containing disparities was generated. A snapshot of the output stream and the corresponding events at the same time are visualized in Figure 3. The colors of the pixels of the output stream in Figure 3b correspond to the depth of the particular associated event. Note that the depths have not been calibrated for our particular configuration of the sensors, and thus the corresponding values for the depths are inaccurate.

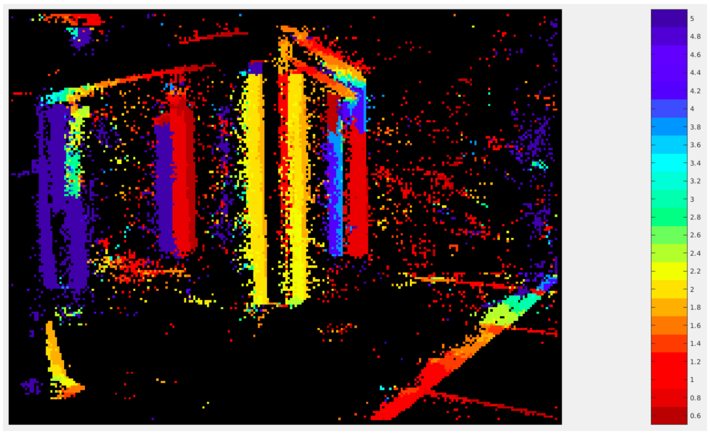
3.2 Algorithm implementation

The times required for the 'MATLAB', 'MATLAB (modified)' and 'C++' implementations to process the 'box1', 'walking2' and 'rotate' event streams are shown in Table 2. As the number of events increased in the streams, each of the algorithms runtime increased. In particular, Table 3 shows that the runtimes of the 'MATLAB (modified)' and 'C++' implementations increases almost linearly with the number of matched events. Also, across all three datasets, it was found 'C++' had the lowest runtime, followed by 'MATLAB (modified)' and finally 'MATLAB'.

Table 2: Stereo matching processing runtimes on event streams

Implementation	box1 (secs)	walking2 (secs)	rotate (secs)
MATLAB	230	2319	16379
MATLAB (modified)	224	1743	8387
C++	37	282	1389


(a) Stereo events from left and right sensors.



(b) Estimated depth for stereo-matched events.

Figure 3: Comparison between stereo events and stereo-matched events at the same time.

Table 3: Number of stereo-matched events processed per second

Event stream	MATLAB	MATLAB (modified)	C++
`box1`	132	136	825
`walking2`	107	143	884
`rotate`	67	132	799

4 Discussion

4.1 Algorithm precision

In order to verify that the combination of the pipeline and algorithm was operational, we looked at the visualization of the output for 'rotate' (Figure 3). Although the program for visualization was not calibrated to calculate depth according to our particular hardware setup, we were less concerned with obtaining accurate distances compared to making sure that the estimated disparities make sense. Despite not recording ground truth data, we can attempt to verify success of the pipeline by comparing snapshots of the stereo events and depth estimations at the same time. Focusing on the bottom right corner of the matched events snapshot, we observe that there is a diagonal line of events (Figure 3b). The colors of the events indicate that the bottom of the diagonal is closer to the sensor (red), and the top of the diagonal is farther (blue). In terms of disparity, this means that events at the bottom of the diagonal should have a higher disparity than the events at the top. It is clear that by overlapping the two stereo images (Figure 3a), there is more disparity between the corresponding events at the bottom of the diagonal than at the top, thus agreeing with the estimated depth. This supports the idea that the events are being properly processed through the pipeline to interface with the algorithm. On the other hand, some estimates seem to be noticeably inaccurate. This is most obviously seen when comparing the front and back leg of the table (Figure 4). The corresponding depth estimates (Figure 3b) are incorrect, since parts of the back leg are estimated to be closer than front leg. This is believed to have occurred as the events in the rows corresponding to the legs of the table seem to be distributed with high variance (since there are many leg chairs), compared to the rows containing the diagonal. Therefore the inaccuracies would have occurred due to the algorithms being unable to handle the nature of the events, as opposed to errors in the pipeline. Note that this analysis was non-rigorous and mainly based on visual inference. In order to rigorously confirm this, it is suggested that recordings with and without a high variance of events distributed across the rows are processed, also measuring the ground truth depth to allow for comparison of accuracy [Xie et al., 2017]. Furthermore, it is recommended that stereo-rectification is also implemented in the processing pipeline, since it could be that many of the rows are not aligning correctly for matching.

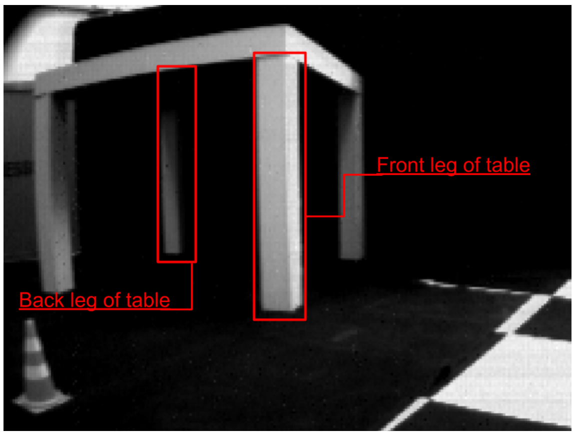


Figure 4: Snapshot of image frame with labels for front and back legs of table

4.2 Algorithm runtime

There are two major and one minor factor which influence the runtime of the algorithm. The first major factor is the programming language the algorithm is implemented in. The C++ implementation beats the MATLAB implementation by a factor of 6. The second major factor is the number of stereo-matched events. As stated before in section 3.2, the runtime increases almost linearly with the number of matched events. This makes estimating the runtime of new data sets more difficult, since the number of matching events is unknown prior to running the algorithm. In contrast the number of input events is known, which leads to the minor factor on runtime. While the algorithm is efficient in discarding events, which have no match, it cannot do so instantaneously. Therefore, the runtime slightly increases with the number of discarded events. This can be shown with the C++ implementation results. First the 'box1' event stream is defined as a baseline, with 825 stereo-matched events processed per second and 31% of its events matching. In comparison, in the 'rotate' event stream only 3.7% of the events match, but still 799 stereo-matched events are processed per second, which is only a decrease of 3.2%. In the same manner, it can be shown that the 'walking2' event stream has a 7.2% increase of stereo-matched events per second, with 39% of its events matching. In conclusion, the number of input events alone is an unfit indicator to predict the algorithms runtime. A better prediction could be achieved by estimating the number of matching events.

5 Conclusion

Initially, the goal was to establish a data processing pipeline to allow easy interfacing between a local stereo sensor rig and preexisting event-based stereo matching algorithm based on belief propagation. This was believed to have been successful despite the output containing regions where depth estimations do not make sense, since events are distributed with high variance in regions where inaccurate estimates occurred. This is likely to be caused by the algorithm as opposed to the pipeline, but should be confirmed via further testing, considering ground truth as well as rectification of events. Attempts were also made to speed up the preexisting stereo matching algorithm by modifying the code in MATLAB, as well as reimplementing it in C++. It was found that both of these versions scale linearly with matched events, and the C++ version performed 6 times faster than the MATLAB implementation.

Acknowledgement

Thanks to Lukas Everding for supervising the project and Matthias Emde for assisting with the hardware setup.

References

[Besse et al., 2012] Besse, F., Rother, C., Fitzgibbon, A., and Kautz, J. (2012). PMBP: PatchMatch Belief Propagation for Correspondence Field Estimation.

[Brandli et al., 2014] Brandli, C., Berner, R., Minhao Yang, Shih-Chii Liu, and Delbruck, T. (2014). A 240 × 180 130 dB 3 μs Latency Global Shutter Spatiotemporal Vision Sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341.

[CamuÃ±as-Mesa et al., 2014] CamuÃ±as-Mesa, L. A., Serrano-Gotarredona, T., Ieng, S. H., Benosman, R. B., and Linares-Barranco, B. (2014). On the use of orientation filters for 3D reconstruction in event-driven stereo vision. *Frontiers in Neuroscience*, 8:48.

[Conradt et al., 2009] Conradt, J., Berner, R., Cook, M., and Delbruck, T. (2009). An embedded AER dynamic vision sensor for low-latency pole balancing. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 780–785. IEEE.

[Czech and Orchard, 2016] Czech, D. and Orchard, G. (2016). Evaluating noise filtering for event-based asynchronous change detection image sensors. In *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 19–24. IEEE.

[Firouzi and Conradt, 2016] Firouzi, M. and Conradt, J. (2016). Asynchronous Event-based Cooperative Stereo Matching Using Neuromorphic Silicon Retinas. *Neural Processing Letters*, 43(2):311–326.

[Kogler et al., 2011] Kogler, J., Humenberger, M., and Sulzbachner, C. (2011). Event-Based Stereo Matching Approaches for Frameless Address Event Stereo Data. pages 674–685.

[Lagorce et al., 2017] Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., and Benosman, R. B. (2017). HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1346–1359.

[Lichtsteiner et al., 2008] Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128$\times$128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576.

[Lucas and Kanade, 1981] Lucas, B. D. and Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision (DARPA). In *Proceedings of the 1981 DARPA Image Understanding Workshop*, pages 121–130.

[Mustafah et al., 2012] Mustafah, Y. M., Noor, R., Hasbi, H., and Azma, A. W. (2012). Stereo vision images processing for real-time object distance and size measurements. In *2012 International Conference on Computer and Communication Engineering (ICCCE)*, pages 659–663. IEEE.

[Rogister et al., 2012] Rogister, P., Benosman, R., Sio-Hoi Ieng, Lichtsteiner, P., and Delbruck, T. (2012). Asynchronous Event-Based Binocular Stereo Matching. *IEEE Transactions on Neural Networks and Learning Systems*, 23(2):347–353.

[Xie et al., 2017] Xie, Z., Chen, S., and Orchard, G. (2017). Event-Based Stereo Depth Estimation Using Belief Propagation. *Frontiers in Neuroscience*, 11:535.