Low-latency monocular depth estimation using event timing on neuromorphic hardware

Stefano Chiavazza
Neuromorphic Computing Lab
Intel Labs
chistefano@gmail.com

Svea Marie Meyer
Neuromorphic Computing Lab
Intel Labs
svea.marie.meyer@intel.com

Yulia Sandamirskaya
Neuromorphic Computing Lab
Intel Labs
yulia.sandamirskaya@intel.com

Abstract

Depth estimation is an important task in many robotic applications. It is necessary to understand and navigate an environment and to interact with objects. Different active and passive sensing solutions can be used for depth estimation, with different tradeoffs in accuracy, range, latency, dealing with challenging light conditions, power efficiency and price. Event-based dynamic vision sensors (DVS) are particularly well-suited for situations in which low latency and low power vision are needed, e.g. *on a fast mobile robot. In this work, we present an event-based depth estimation method with a single DVS using a novel depth from motion algorithm targeting neuromorphic hardware. The system first computes the optical flow on the neuromorphic chip and then computes the depth by combining optical flow with the camera velocity. The method assumes only translational motion and it successfully reconstructs the depth from the measured flow. The method can achieve low-latency depth estimation (<0.5ms) while maintaining a small network size, allowing for better scalability. We tested the algorithm on Intel's neuromorphic research chip Loihi 2.*

1. Introduction

Depth estimation is an important task in many robotic applications. It is necessary to estimate distances to objects in an environment and is an important component of the problem of simultaneous localization and mapping (SLAM). Different visual methods for depth estimation include active sensing using lasers (lidar) or structured light setups, passive sensing using multi-view geometry (stereo) or monocular camera solutions using depth-from motion or parallax, as well as deep neural networks trained on annotated image-depth datasets. All methods have their pros and contras, but the passive camera-based solutions often win due to their simplicity, low cost and applicability both in indoor and outdoor settings. However, in challenging scenarios, camera-based solutions may fail. For instance, fast motions often cause motion blur in the images, making depth estimation harder. The fixed sampling rate of the cameras also makes them less suited for latency-critical applications. Furthermore, they often fail to produce good images in low light conditions and have a relatively low dynamic range. Event-based dynamic vision sensors (DVS), on the contrary, deal well with these scenarios [12]. Here, we propose a novel method to efficiently compute depth from a moving event-based camera using neuromorphic hardware.

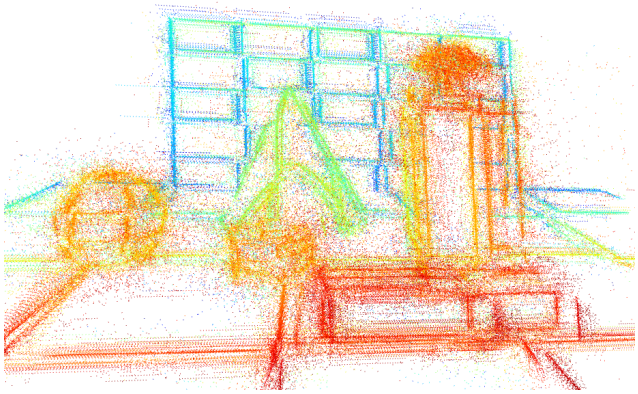


Figure 1. Depth measurements over a 4 seconds period reprojected in 3D space.

Event cameras are bio-inspired vision sensors that feature asynchronous pixels. They do not work at a fixed sam-

pling rate like conventional cameras. Instead, every pixel independently detects brightness changes. When a large enough change is detected, an event is triggered and sent out to the computer. Each event carries the address of the pixel it was emitted from. The DVS output then consists of an events stream, representing local brightness changes. This output is sparse and informative, offering several advantages over traditional cameras: event cameras offer lower power consumption, low latency, and high dynamic range. These are all important characteristics for robotics applications.

Conventional image processing techniques cannot be easily used on event data, since there are no images or frames leaving the sensor. Image processing methods are not designed to take advantage of the asynchronous and sparse characteristics of the DVS. New approaches are required to take full advantage of these sensors.

Several novel methods have been proposed to process events and extract relevant information. The majority of these run on conventional hardware: CPUs or GPUs. This usually leads to bottlenecks as traditional hardware is not designed to handle the sparse output of event cameras. Events are usually accumulated onto frames and then processed as a batch. In this case, algorithms are typically not designed to process each incoming event individually.

Hardware specifically designed to handle sparse and asynchronous data is necessary to take full advantage of the event cameras. Neuromorphic processors use a computational model that leverages the same concepts of sparse and asynchronous communication. They receive, send, and process information with the use of spikes in a similar fashion to event-based signal stream of the DVS (and similar to how our eye samples and our brain processes visual information). Both analog and digital neuromorphic platforms have been developed, each with its advantages. Some of such neuromorphic platforms are: Spinnaker [11], IBM TrueNorth [20] and Intel's neuromorphic research chips Loihi 1 and 2 [7] [8].

In this work, we use the Loihi 2 chip. It allows the definition of custom neuron processes and it also introduces the possibility to send graded spikes between neurons. These features allow us to have a simpler network that uses fewer spikes for the communication between neurons.

2. Related Work

Previous research on monocular depth estimation using event-based cameras focuses on methods that run on conventional hardware, such as CPU and GPU. The work in [19] estimates the 3D scene structure and the 6-DoF camera motion from a single event camera. The method runs on a standard PC with a GPU and it can process up to 1M events per second. This is enough for slower camera motions that trigger fewer events, but it is not able to run in real-time with very high event rates.

The EMVS method [25] leverages the sparsity and the high temporal resolution of event cameras. It can reconstruct a sparse depth map of the environment from the event stream and the camera groundtruth position. Their implementation can process 1.2 million events/s on a single core on a standard laptop CPU. Similarly, [13] uses contrast maximization to find the best depth value that fits the event stream and a given camera motion.

More recent methods use ANNs to predict monocular depth from event data [14, 18]. In this work, simulated data is used to train the network. The data was recorded using the CARLA simulator [9]. The simulator provided a groundtruth depth that was used to supervise the training.

Few methods explore the use of neuromorphic chips to extract depth information from the event stream on the chip directly. The work in [16] determines the distance of objects by varying the camera focus and observing the appearance of the objects with an event camera. They implemented the method in simulation and they were able to reconstruct the structure of the scene.

More work has been done on optical flow estimation on neuromorphic hardware from a stream of events. The work in [23] was one of the first methods to compute optical flow from event cameras with a spiking neural network. The authors implemented coincidence detectors using synaptic delays creating units sensitive to different speeds. The downside of this method is that it can only measure the optical flow at discrete pre-defined velocities.

The works in [1, 3, 24] define a biologically plausible architecture for estimating the optical flow. In [24] a 3-layer structure was used, inspired by the stages of visual processing in biological brains. The authors implemented a set of spatio-temporal oriented filters to process the event input. The response of each filter was combined and used to estimate the magnitude and direction of the optical flow.

The work in [10] uses the same energy-based approach but is implemented on hardware and takes advantage of the characteristics of the Loihi 2 chip using resonate-and-fire neurons. These methods are good at estimating the flow, especially in areas of the image with complex textures. One downside is that they require a larger number of events to obtain an estimate compared to other methods [1].

In [15, 17, 21], a correlation-based approach was implemented on neuromorphic hardware. In these works, the same *time-to-travel* approach was used. They measure the time it takes for an edge to travel from one pixel to the next, the correlation is established thanks to the high temporal resolution of event cameras.

In [15], the authors use a mixed analog/digital neuromorphic processor implementing a network that measures the *time-to-travel* using LIF neurons. The work in [17] uses the same approach, but they implemented it on the IBM True

North neuromorphic processor [20].

The work in [21] uses a similar approach but with a different implementation. The output spike rate here is not constant like in [15], it depends on the time measured. Unlike before, a neuron only starts spiking when the second input spike is received. Conceptually similar work in [5] measured *time-to-travel* on a microcontroller. This allowed the researchers to encode the information as a fixed-point integer, making the optical flow easier to process. On the other hand, a microcontroller is not suited to process all the parallel information coming from an event camera.

Most methods for monocular depth estimation take into consideration events from a larger time window. On the other hand, measuring the instantaneous depth usually requires two or more cameras [12]. Furthermore, the computation is usually performed on CPU or GPU, introducing a latency. Our method focuses on low-latency and low-power depth estimation running on neuromorphic hardware.

3. Method

The presented network consists of two layers: the optical flow layer and the depth estimation layer. The optical flow is obtained by measuring the time-to-travel: the time it takes for an edge to move between pixels of the visual array. The general approach is inspired by the Time Difference Encoder (TDE) from [21]. The implementation, however, is significantly different.

The TDE in the original work encodes the time-to-travel, or more generally the time difference between two spikes, into a rate-coded spike output. By measuring the time difference between events in adjacent pixels, it is able to determine the direction and velocity of a moving edge.

The method presented in this work measures the same time-to-travel but encodes it using graded spikes. Typically, spikes carry only binary information, but on Loihi 2 spikes can also carry a value. We use this feature to send the measured time difference as a single spike, allowing for efficient encoding and low latency. This results in fewer spikes being sent and lower latency compared to rate encoding. The time difference is then used together with the camera velocity, which is assumed to be known, to calculate the distance to objects in the scene.

3.1. Normal flow computation

Due to the aperture problem, we can't measure the optical flow directly by observing a moving edge. Additional information is necessary to estimate the full optical flow. What we measure instead is the normal flow, which depends on the orientation of the edge.

A single time difference unit can measure the time-of-travel in a single direction. 4 units are combined together to obtain an x-y normal flow measurement.

3.1.1 Motion Detector in one direction

Similarly to the TDE in [21], a single motion-sensitive unit takes two inputs: a facilitating and trigger spikes. The unit measures the time between the two spikes and is sensitive to motion in a single direction. The facilitating spike has to come before the trigger, otherwise, no output is produced.

The Loihi 2 chip allows us to define custom neuron models defined in microcode. We can use this feature to implement a model that measures the time difference between the two inputs. The model is implemented as a simple counter that starts when the facilitating spike is received and stops once the trigger is received. The value of the counter is then sent as a graded spike. Additionally, the model also checks that the polarities of the two spikes match. The logic is defined in Algorithm 1.

Algorithm 1 Simple logic of a motion detection unit

$trigger \leftarrow getTrigger()$
$facilitating \leftarrow getFacilitating()$
$counter \leftarrow getCurrentCounter()$
$polarity \leftarrow getStoredPolarity)$
if $trigger$ & $polarity(trigger) = polarity$ **then**
  $sendSpike(counter)$
  $counter \leftarrow 0$
end if
if $counter \neq 0$ **then**
  $counter \leftarrow counter + 1$
end if
if $facilitating$ **then**
  $counter \leftarrow 1$
  $polarity = polarity(facilitating)$
end if

3.1.2 A full time-of-travel unit

A full unit capable of measuring the optical flow in x-y is made by combining 4 motion detectors. Each detector is sensitive to a different direction. Each pixel has a full unit centered around it. Figure 2 shows the structure of a single unit. As mentioned before, the measurement is affected by the aperture problem thus only the normal flow can be measured.

Each motion detector is sensitive to a single direction. When an opposite direction is presented to it, the trigger spike will receive an event before the facilitating. This does not result in any spikes. Only when the motion matches the direction the unit is sensitive to, an output is produced.

Noise and complex textures could cause more than two motion units to spike at the same time. In that case, there could be an inconsistency, *e.g.* both a leftward and rightward motion present at the same time. To solve it, we apply a simple heuristic. When such inconsistencies are detected,

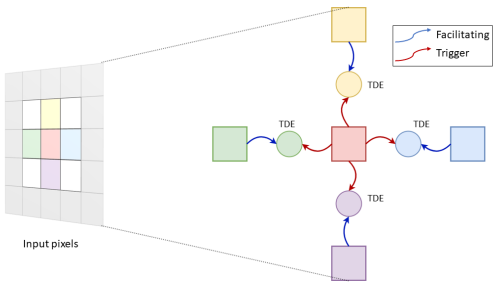


Figure 2. Structure of a full motion sensitive unit. The input pixels are represented with squares, while the time-of-travel units – with circles. The trigger input is the same for all 4 units, while the facilitating input depends on the direction.

the smaller time difference is kept and the larger one is discarded.

In general, each event produces two measurements. A vertical and a horizontal time-of-travel. This is true as long as other events have been received previously in the neighboring pixels. At least two events are needed to measure a time difference and consequently to measure depth.

3.2. Depth estimation

There are a couple of key assumptions that we have to make in order to measure the depth of the scene: (1) The scene is static, without dynamic objects; (2) The camera moves with a known velocity; (3) The camera is only subject to translation and no rotation. The relationship between the optical flow, the scene structure and the camera velocity is well known [4, 6], and it is given by:

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \frac{1}{Z}\boldsymbol{At} + \boldsymbol{B\omega}, \tag{1}$$

where $\dot{u}$ and $\dot{v}$ represent the horizontal and vertical components of the optical flow, Z is the depth at the considered position, $\boldsymbol{t} = \begin{pmatrix} t_x & t_y & t_z \end{pmatrix}^\intercal$ is the translating camera velocity and $\boldsymbol{\omega} = \begin{pmatrix} \omega_x & \omega_y & \omega_z \end{pmatrix}^\intercal$ is the rotational camera velocity,

$$A = \begin{pmatrix} -f & 0 & x \\ 0 & -f & y \end{pmatrix} \tag{2}$$

and

$$B = \begin{pmatrix} \frac{xy}{f} & -(f + \frac{x^2}{f}) & y \\ f + \frac{x^2}{f} & -\frac{xy}{f} & v \end{pmatrix}. \tag{3}$$

Here, f is the focal length of the camera and x, y are the pixel position.

The normal flow is defined as the projection of the optical flow onto the direction normal to the edge. Let's define the vector $\boldsymbol{u}$ as the 2D optical flow and the unit vector $\hat{\boldsymbol{n}}$ as the normal direction. Projecting $\boldsymbol{u}$ onto $\hat{\boldsymbol{n}}$ is equivalent to taking the dot product, so the normal flow magnitude is equivalent to: $|\boldsymbol{n}| = \hat{\boldsymbol{n}} \cdot \boldsymbol{u}$. See [2] for more details.

By multiplying both sides of Eq. (1) by $\hat{\boldsymbol{n}}$ we get:

$$|\boldsymbol{n}| = \frac{1}{Z}\boldsymbol{At} \cdot \hat{\boldsymbol{n}} + \boldsymbol{Bw} \cdot \hat{\boldsymbol{n}}. \tag{4}$$

By rearranging we can solve for Z:

$$Z = \frac{\boldsymbol{At} \cdot \hat{\boldsymbol{n}}}{|\boldsymbol{n}| - \boldsymbol{Bw} \cdot \hat{\boldsymbol{n}}}. \tag{5}$$

The flow computed previously, however, is not a velocity in pixels/s. It is a time difference representing the time it takes for the edge to move by one pixel. Converting the time difference to the velocity requires a value inversion. Given the fixed point representation available on Loihi 2, this is not easily done on the chip. Instead, we can reformulate the solution to make use of the time difference directly and avoid doing more complex computations.

3.2.1 Computing depth with the time difference

For a more detailed analysis on optical flow measurement using event timing, see [3]. In short, the 2D time difference is a vector with the same direction as the normal flow: $\hat{\boldsymbol{n}} = \hat{\boldsymbol{td}}$, but the magnitude is the inverse: $|\boldsymbol{n}| = \frac{1}{|\boldsymbol{td}|}$. Where $\boldsymbol{td}$ is the time difference vector measured.

We substitute these in Eq. (5) and get:

$$Z = \frac{\boldsymbol{At} \cdot \hat{\boldsymbol{td}}}{\frac{1}{|\boldsymbol{td}|} - \boldsymbol{Bw} \cdot \hat{\boldsymbol{td}}}. \tag{6}$$

We also know that: $\hat{\boldsymbol{td}} = \frac{\boldsymbol{td}}{|\boldsymbol{td}|}$:

$$Z = \frac{\boldsymbol{At} \cdot \boldsymbol{td}}{1 - \boldsymbol{Bw} \cdot \boldsymbol{td}}. \tag{7}$$

This definition uses the time difference directly, but still has a division that we would like to avoid. The translational motion is necessary to estimate the depth of the scene, but the rotational motion is superfluous. We assume then that the camera doesn't have any rotational motion. This way the solution can be simplified to:

$$Z = \boldsymbol{td} \cdot \boldsymbol{At} \tag{8}$$

We can assume that the vector $\boldsymbol{At}$ is known, as it will be computed off-chip and depends purely on the camera motion. The solution then consists of a simple dot product, which involves only multiplication and addition that can be easily computed on Loihi.

4. Implementation

The method is implemented on Loihi 2 and uses graded spikes and custom neuron models. A single Loihi 2 chip is made up of multiple (128) neuromorphic cores. Each core updates the internal values for some neurons and handles the computation and transmission of spikes.

In order to fit all the neurons necessary for the depth estimation, we divide the input image into patches. The dimension of a patch is chosen to fit inside a single neuromorphic core. Each core then handles the events coming from that patch and the computation associated with it.

The network can be seen as a two-layer network. One layer measures the time-of-travel and the second one estimates depth. Half the cores implement one layer while the other half implements the other.

The method assumes that an edge triggers only one event per pixel. This makes it easier to establish the correspondence between events and measure the time difference. To enforce this, we use a refractory period on the input events. This could be implemented as part of the network, but we implement this as pre-processing on the input events.

4.1. Time-of-travel measurement

The model for the time difference measurement is relatively simple, it is defined in microcode and follows the logic expressed in Algorithm 1. Each neuron only stores two values: a *counter*, and a *sign*. The first one keeps track of the time between spikes. The second stores the polarity of the last event received.

Each unit has two inputs: the facilitating input and the trigger input. This means that each one requires 2 accumulators to store the incoming spikes. An accumulator is responsible for storing the input values for a neuron. In this case, we need to distinguish between two different inputs and we need two different accumulators.

This is important as each core has a limited number of accumulators available. This determines the dimension of the patch that each core can handle.

Other solutions are also possible. Weighted inputs could be used for the distinction between inputs. This could reduce the number of accumulators needed making the network smaller.

4.1.1 Timestep and synchronization

So far we have discussed how the time measurement is done by increasing a counter value at each timestep. This assumes that the updates happen at a constant rate. Normally the neuromorphic cores implement a barrier synchronization mechanism that keeps them operating at the same time scale. In other words, a neuron does not start the next timestep update until all the relevant neurons have also been updated.

This mechanism alone does not guarantee a constant timestep duration. It is possible to use the embedded x86 on Loihi to advance the computation at a constant rate. For our tests, we used a timestep duration of 1ms. This effectively introduces a small discretization in the optical flow measurement and in the depth estimate. See Sec. 5 for more details. In our implementation, the input is sent to the network at a constant rate. This provides constant timestep progression.

4.1.2 Full time of travel unit

A single time-of-travel unit can only measure the time difference in one direction. To compute the flow, four units are necessary, one for each direction: up, down, left, and right. Figure 2 shows the structure of a single unit. There is one such unit centered around each pixel. In other words, for each pixel, we have 4 time-of-travel units, each with 2 inputs requiring two accumulators.

In other words, measuring the time difference requires 8 accumulators per pixel. On a single core, we can have up to 8192 indexable accumulators. This means that a single core can take care of 1024 pixels, or a patch of 32x32. For simplicity, we divide the input into patches of 30x30, which divides nicely the chosen input resolution of 240x180 px. In total, 48 cores are needed to implement the flow layer for the mentioned input resolution.

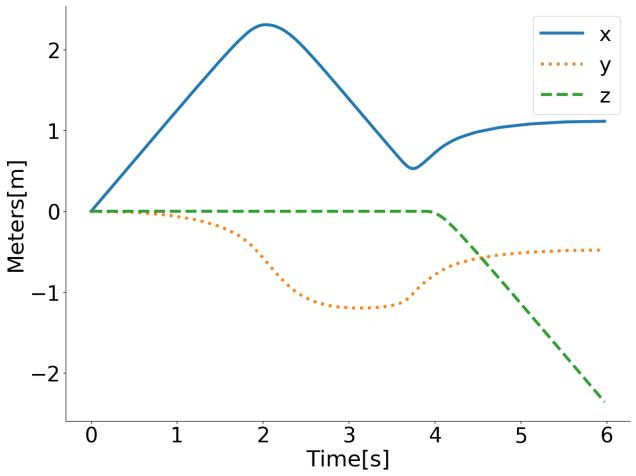
When the trigger pixel receives an event, all 4 units (if they have an active counter) send the current counter value as a graded spike to the corresponding neuron in the depth estimation layer.

4.2. Depth estimation

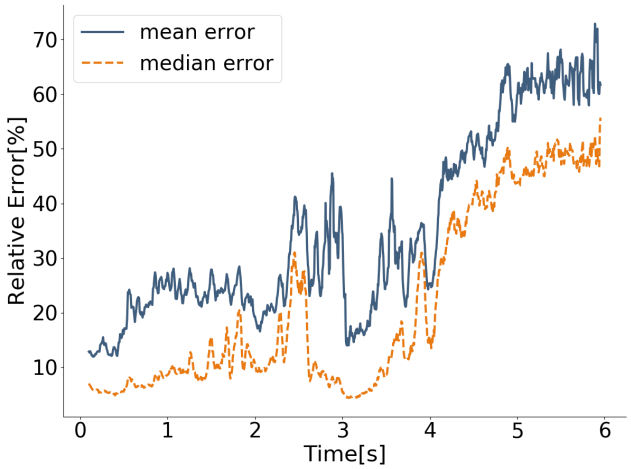
The same idea of dividing the image into patches applies to the depth estimation layer. The neuron model used in the depth layer is more complex and requires more resources compared to the previous model. There is, however, only one neuron for each pixel. This, overall, reduces the total number of resources required for the depth estimation layer compared to the flow layer. As a result, larger patches could fit inside a single core, but for simplicity, we use patches of the same dimension as before: 30x30. Implementing the depth estimation layer requires then another 48 cores for a total of 96 cores. This means that with the input resolution of 240x180, the network can fit on a single chip.

Each neuron in this layer has 6 inputs in total. 4 are for the time difference measurements from the previous layer and 2 are for the known camera velocity. This is the $\boldsymbol{At}$ term in Eq. (8). The other term is the time difference measured.

In an ideal situation, only two of the four directions will output a spike, one horizontal and one vertical. However, due to noise or multiple edges close to each other, all four units could send a spike. In other words, at the same time,



(a) Plot of the camera position over time.



(b) Plot showing the mean and median estimation errors over the whole sequence.

Figure 3. Plots of the camera position and the depth error over time.

we could have both a leftward and rightward motion. We apply a simple heuristic to solve this conflict. When two values are received, only the smaller one is considered. This reduces the time differences to only two measurements. The model then simply performs the dot product of Eq. (8) which can be easily implemented with multiplication and addition operations.

Note how the value of $\boldsymbol{At}$ depends on pixel position. When the camera motion comprises only of horizontal and vertical motion, the value is the same for every image position. However, when it moves along the z-axis, each pixel would have to receive a different position.

This value is updated at fixed time intervals with the ground truth position of the camera. The value is stored inside the neuron to have it available at every timestep. This means that when the velocity is updated, a spike would need to be sent for every pixel at the same time. This is somewhat inefficient and could cause occasional slowdown. To reduce the number of spikes sent, the image area is divided into patches that share the same value for $\boldsymbol{At}$. This does not affect the quality of the results when there is only motion along the x and y-axis. It has a minor impact when the camera is moving along the z-axis. However, in general, the motion along x and y is more important for depth estimation.

4.3. Input and Output

Currently, the method is not running with a live camera input. The event data is pre-loaded on the embedded cores and then fed to the neuromorphic cores during execution. The embedded core is a small x86 processor present on the Loihi chip. It takes care of various tasks, in this case it is used to send data to the neuromorphic cores. The input method is not optimized for event data. The input data is stored on the embedded core and more memory than necessary is reserved. This could cause some slowdowns. See Sec. 5.1.1 for more details.

Once a depth measurement is computed, the graded spike output from the network can be used to carry out further computation on the chip. In the current implementation, the output cannot be streamed off-chip. The embedded core cannot handle a large number of graded spikes and stream them directly to a host machine. More optimized methods for I/O should make it possible to stream the spikes with depth information and stream the event data more efficiently. At the moment, software support for the chip limits the I/O performance.

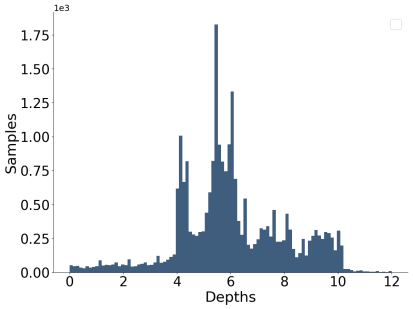
Currently, to analyze the result of the network, the computation has to be paused to probe the internal values. This slows down the execution significantly. When the values are not probed, the method can run in real-time.

5. Experiments

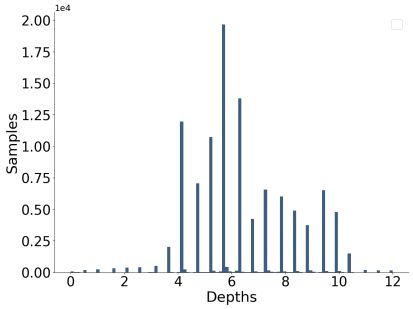
We tested the method on two datasets. A synthetic dataset where the camera has a more general motion pattern. And on real data using the *slider-depth* dataset from [22]. This dataset, however, includes only a horizontal camera motion. The datasets we used present very little lens distortion, so no corrections were applied to the event data.

5.1. Synthetic Data

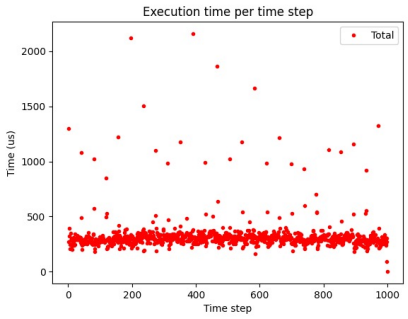
The synthetic data is generated with the eSim event simulator [26]. Given a virtual 3D scene and the camera motion, it generates the stream of events and the ground-truth depth data. We use the ground-truth depth to measure the estimation error.



(a) Camera moving at $0.55m/s$



(b) Camera moving at $2.6m/s$



(c) Timestep duration: the execution time for each timestep when the duration is not kept constant by the embedded core.

Figure 4. Histograms of depth measurements for two different camera velocities on the same scene.

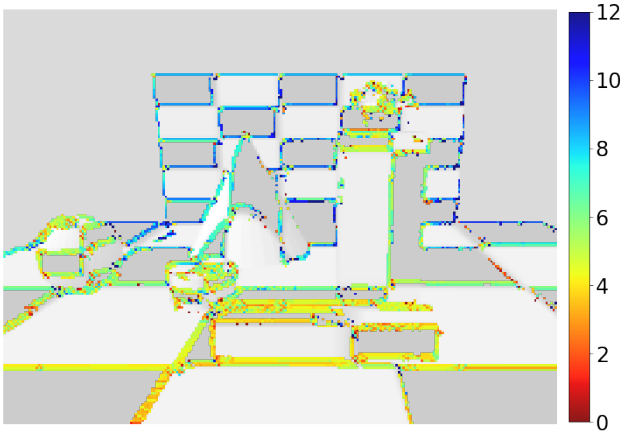


Figure 5. Output depth measurements, accumulated over 70ms. The values are in meters

Mean[m]	Relative[%]	Median[m]	Relative[%]
1.670	31.937	0.854	16.926

Table 1. Depth estimation accuracy on the synthetic dataset with camera movement along all axis.

Mean[m]	Relative[%]	Median[m]	Relative[%]
0.810	13.279	0.345	5.598

Table 2. Depth estimation accuracy with the camera moving horizontally.

Figure 3a shows the camera trajectory used in the synthetic sequence. The portions of the sequence where the camera motion is mostly horizontal or vertical are simpler and yield better results. Figure 3b show the relative errors over the whole sequence. The method relies on the assumption that during a single timestep, the velocity is constant. Sudden changes in direction make the estimate worse.

The relative error plot also shows that the error is lowest when the camera is moving almost exclusively horizontally. When the camera moves along the z-axis the depth estimate can be very imprecise. This is likely caused by the slow optical flow generated by such a motion. Objects in the center of the field of view will appear almost static, making the optical flow measurement very imprecise. Figure 5 shows the depth measurements accumulated over a period of 70ms.

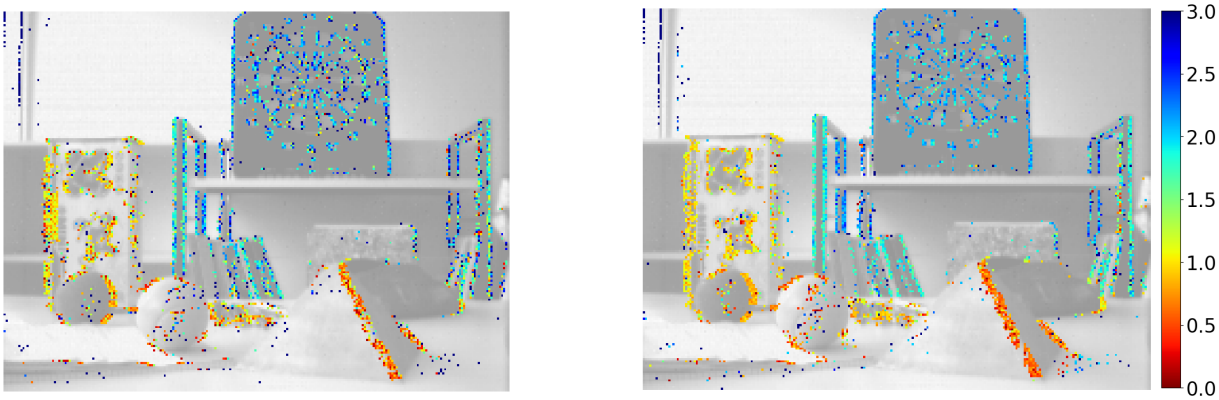
Table 1 shows the mean and median errors over the whole sequence. This includes the high error at the end of the sequence. Further, we run the method on a second simulated sequence that has only a horizontal camera trajectory. Table 2 shows the measurement errors for this scenario.

5.1.1 Timestep duration choice

The histograms in Figs. 4a and 4b show all the depth values measured with the camera moving horizontally for two different sequences. In the first one, the camera was moving at a total speed of $0.55m/s$. In the second, the camera moves at a higher speed of $2.6m/s$. From the histograms, it is clear how the results from the fast-moving camera are more discretized. The high velocity was chosen to show its impact on the results.

This is a consequence of time discretization, which depends on the timestep duration chosen for the computation. It also affects the maximum depth that can be measured since the time difference counters have limited precision. Slow movements of distant objects, cause the counters to overflow, limiting the maximum time measurable. In the current implementation, we are using 8 bits for the counter representation. It is possible in theory to use more, but we found that for the range of depths and camera velocities used in this work, this precision works well.

If we don't use the embedded core to control the time



(a) Time difference measured between adjacent pixels

(b) Time difference measured between events 2 pixels apart.

Figure 6. Accumulated depth estimates over 30ms for the *slider-depth* sequence. Values are in meters.

progression, we can let the cores run as fast as possible and observe the duration of the computation at each timestep. Figure 4c shows the total duration of each timestep. The time includes the state update of the neurons and the spike handling. The execution time at each timestep is around $400\mu s$. There are some timesteps where the computation takes longer, this is likely due to the inefficient methods used for sending the event data and the camera velocities to the chip. At the moment, the input is limited by the software support for the chip[1].

Overall, the duration is almost always under the target timestep duration of 1ms. This shows that it would be possible to target smaller timesteps, but it would require further optimization of the data input. A shorter timestep duration would also increase the precision of the flow measurement. It would, however, require a higher precision for the time difference counters. It would also increase the activity of the chip, increasing the power draw. When running the network on half the input resolution we notice a decrease in the average execution time. The majority of the timesteps take around $100\mu s$. When running at half the resolution, we maintain the same number of neurons per core, but we reduce the total number of neuromorphic cores used. Again, we believe the higher execution time in the previous example to be caused by the input method used. Further testing is needed to characterize the timesteps duration when better I/O support is available.

The timestep duration also determines the latency of the measurements. In our tests, the $1ms$ timestep would lead to effectively a $1ms$ latency. The latency, however, cannot be measured due to the I/O method used (see Sec. 4.3). The current implementation could be used with $500\mu s$ timestep and latency. Better I/O support could lead to even smaller latencies.

[1]All experiments run on a machine with Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz x8, 32GB RAM running Ubuntu 20.04.2 LTS, python 3.9.7, and v.2.2.0 of the Intel NxCore in Loihi 2 hardware. All performance measurements are based on testing as of March 2023 and may not reflect all publicly available security updates. Results may vary.

5.2. Real Data

To test the method on real-world data, we use the slider-depth sequence from [22]. It consists of a camera moving along a controlled slider. This gives an accurate groundtruth position of the camera from which we can extract the velocity. The scene presents objects placed at various distances, it does not however provide any depth information.

Figure 6a shows the depth measurements accumulated over 30ms. The results from the real-world data contain a significantly larger amount of noise. Figure 6b shows the same scene, but the network is slightly modified to measure the time difference between events two pixels apart. The measurements appear to be more consistent, but not by much. Measuring the time difference between larger distances improves the precision, but increases the chances of false correspondences between events, leading to errors.

This is a problem also when measuring the time difference between adjacent pixels. The method works well for straight edges, but it becomes less reliable when observing complex textures.

6. Conclusion

We presented a lightweight and low-latency method for depth estimation that can run on the Intel Loihi 2 neuromorphic chip. It can process the data event by event and produce depth measurements with a theoretical latency of 1ms or lower. The method currently has some limitations on the types of motions supported, limited only to translational motion. The overall accuracy is lower when compared to other monocular CPU-based methods that consider larger windows of events. However, it achieves lower latency and power consumption running on neuromorphic hardware.

References

[1] Francisco Barranco, Cornelia Fermuller, and Yiannis Aloimonos. Bio-inspired motion estimation with event-driven sensors. In Ignacio Rojas, Gonzalo Joya, and Andreu Catala, editors, *Advances in Computational Intelligence*, pages 309–321. Springer International Publishing, 2015. 2

[2] Francisco Barranco, Cornelia Fermüller, Yiannis Aloimonos, and Eduardo Ros. Joint direct estimation of 3d geometry and 3d motion using spatio temporal gradients. *Pattern Recognition*, 113:107759, 2021. 4

[3] Tobias Brosch, Stephan Tschechne, and Heiko Neumann. On event-based optical flow detection. *Frontiers in Neuroscience*, 9, 2015. 2, 4

[4] Samuel Bryner, Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. Event-based, direct camera tracking from a photometric 3d map using nonlinear optimization. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 325–331, 2019. 4

[5] J. Conradt. On-board real-time optic-flow for miniature event-based vision sensors. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1858–1863, 2015. 3

[6] Peter Corke. *Robotics, Vision and Control*, volume 118 of *Springer Tracts in Advanced Robotics*. Springer International Publishing, 2017. 4

[7] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018. 2

[8] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A. Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R. Risbud. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021. 2

[9] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 2

[10] E. Paxon Frady, Sophia Sanborn, Sumit Bam Shrestha, Daniel Ben Dayan Rubin, Garrick Orchard, Friedrich T. Sommer, and Mike Davies. Efficient neuromorphic signal processing with resonator neurons. *J. Signal Process. Syst.*, 94(10):917–927, Oct 2022. 2

[11] Steve B. Furber, Francesco Galluppi, Steve Temple, and Luis A. Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014. 2

[12] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, 2022. 1, 3

[13] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3867–3876, 2018. 2

[14] Daniel Gehrig, Michelle Rüegg, Mathias Gehrig, Javier Hidalgo-Carrió, and Davide Scaramuzza. Combining events and frames using recurrent asynchronous multimodal networks for monocular depth prediction. *IEEE Robotics and Automation Letters*, 6(2):2822–2829, 2021. 2

[15] Massimiliano Giulioni, Xavier Lagorce, Francesco Galluppi, and Ryad B. Benosman. Event-based computation of motion flow on a neuromorphic analog neural platform. *Frontiers in Neuroscience*, 10, 2016. 2, 3

[16] Germain Haessig, Xavier Berthelon, Sio-Hoi Ieng, and Ryad Benosman. A spiking neural network model of depth from defocus for event-based neuromorphic vision. *Scientific Reports*, 9(1):3744, 2019. 2

[17] Germain Haessig, Andrew Cassidy, Rodrigo Alvarez, Ryad Benosman, and Garrick 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. 2

[18] Daniel Gehrig Javier Hidalgo-Carrio and Davide Scaramuzza. Learning monocular dense depth from events. *IEEE International Conference on 3D Vision.(3DV)*, 2020. 2

[19] Hanme Kim, Stefan Leutenegger, and Andrew J. Davison. Real-time 3d reconstruction and 6-DoF tracking with an event camera. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 349–364. Springer International Publishing, 2016. 2

[20] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014. 2, 3

[21] M. B. Milde, O. J. N. Bertrand, H. Ramachandran, M. Egelhaaf, and E. Chicca. Spiking elementary motion detector in neuromorphic systems. *Neural Computation*, 30(9):2384–2417, 2018. 2, 3

[22] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM. *The International Journal of Robotics Research*, 36(2):142–149, 2017. 6, 8

[23] Garrick Orchard, Ryad Benosman, Ralph Etienne-Cummings, and Nitish V. Thakor. A spiking neural network architecture for visual motion estimation. In *2013 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 298–301, 2013. 2

[24] Francesca Peveri, Simone Testa, and Silvio P. Sabatini. A cortically-inspired architecture for event-based visual motion

processing: From design principles to real-world applications. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1395–1402. IEEE, 2021. 2

[25] Henri Rebecq, Guillermo Gallego, Elias Mueggler, and Davide Scaramuzza. EMVS: Event-based multi-view stereo—3d reconstruction with an event camera in real-time. *International Journal of Computer Vision*, 126(12):1394–1414, 2018. 2

[26] Henri Rebecq, Daniel Gehrig, and Davide Scaramuzza. ESIM: an open event camera simulator. *Conf. on Robotics Learning (CoRL)*, 2018. 6