Event-based Robot Vision Project Report

Alexandra Zimmer (0412739), TU Berlin, summer semester 2022

October 1, 2022

Abstract

The general objective of the Event-based Robot Vision Project course is to work on a problem in robot perception with the techniques from event-based and computer vision, by investigating and developing tailored algorithms and methods.

The goal of my project is to reconstruct images from event data, via learning a sparse dictionary with atoms consisting of image gradients and corresponding events. At inference time, the algorithm takes only the events as input, and gains the corresponding image gradients from the dictionary. This idea stems from the publication [1], whereas they ultimately use it for direct face detection from events.

1 Project Plan

1.1 Idea

In this project, we implemented one proposed sparse dictionary learning algorithm from [1], which enables reconstructing image gradients from events, from which in turn the images can be reconstructed.

For this purpose, in [1], they develop a patch-based model for the event streams, as a dictionary. It enables modeling event streams with a sparse linear combination of atoms from this dictionary.

As input, the dictionary learning algorithm takes simulated data generated from real images, by simulating moving the image in front of an event camera generating event data and creating the image x and y derivatives (first step in the dictionary learning overview in Fig. 1, which is from [1]). They only simulate linear motion, which is justified by the high temporal resolution of an event camera.

The simulated data is put into a data structure X consisting of image patches and a voxelgrid of number-of-timeslices bins containing the corresponding event data using linear voting in time (second step in Fig. 1), and afterwards the patch-based dictionary is learned from X, to find a sparse signal representation (last step in Fig. 1).

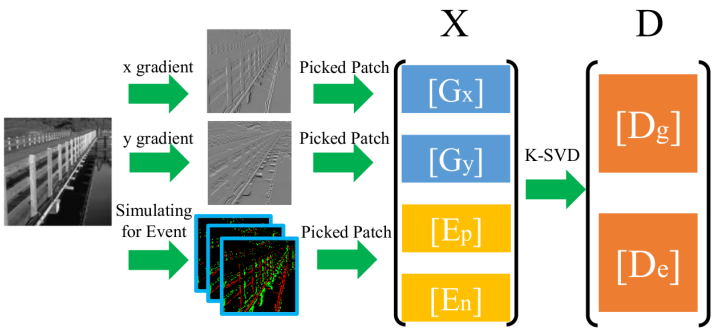

Figure 1: Dictionary learning phase overview

More specifically, this is done via K-SVD, which is singular value decomposition using k-means clustering. The learned dictionary $\mathrm{D} = \begin{bmatrix} D_g \\ D_e \end{bmatrix}$, consisting of D_g as the gradient part and D_e the event part, is found by optimizing Eq. 1, where α is the sparse representation vector, K_s is "the sparsity factor"[1], $||\cdot||_2^2$ is the Frobenius norm and $||\cdot||_0$ is the l0-norm.

$$D = \min||X - D\alpha||_2^2 \quad \text{s.t.} \quad ||\alpha||_0 \leq K_s \tag{1}$$

At inference time, to then reconstruct image gradients from event data, the sparse dictionary representation α is computed for the given set of events, which is then applied to the gradient part

of the dictionary to get the recovered x and y gradients. This means, as indicated by the inference phase overview in Fig. 2 (which is also from [1]), that α is calculated with the use of OMP [5], which is an efficient implementation of K-SVD, taking the positive and negative events Y_e as input, with sparsity factor K_o (Eq. 2).

$$\alpha = \min||Y_e - D_e\alpha||_2^2 \quad \text{s.t.} \quad ||\alpha||_0 \leq K_o \tag{2}$$

The recovered gradient Y_g is calculated by multiplying the sparse representation α with the gradient part D_g of the dictionary ($Y_g = D_g\alpha$), and the recovered images can then be constructed from this via Poisson reconstruction and log inversion.

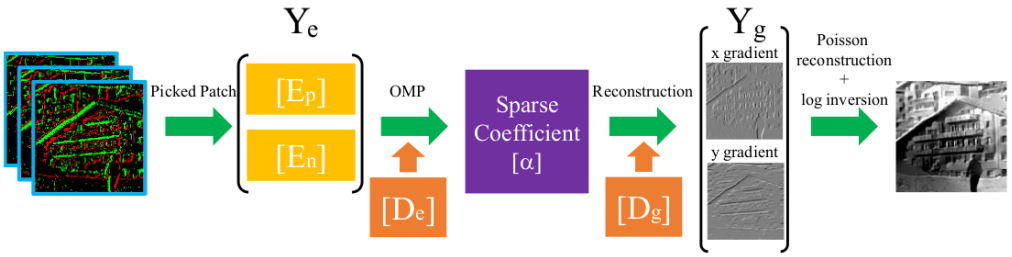

Figure 2: Inference phase overview

The goal is to learn from simulated data of real images, leading to generalization, to apply the dictionary for image reconstruction on other, real event data. In our case, we tried to make the base case work, which is to have one image and its corresponding simulated event data as input for the dictionary learning, reconstruct the image gradients only on the basis of the event data, and compare this to the real image gradients.

1.2 Preconditions

When I started the project, the basic algorithm was already implemented (*base implementation*), but the resulting dictionary and image gradients did not look as expected at all.

This base implementation uses the sample implementation of scikit [4], adapts it and implements the discussed dictionary learning algorithm for events from [1]. It also includes pruning the event data, which is also done in the publication: "Columns with very few events are removed to improve robustness" [1].

One core difference to the aforementioned publication was, that the algorithm only ran on real event data, which does not give the perfect lab conditions as simulated data, but instead can be very noisy, and has non changeable camera motion, etc.

1.3 Concept and Realization

Since the results of the base implementation did not look as they should, the first goal of the project was to get an intuition for what part of the recovered gradient and the learned dictionary looked false and where that might come from, to eventually know how to improve the results.

To understand what dictionaries that are well fitted look like, following material was studied: two Youtube lectures about sparse representation dictionary learning[3], an online tutorial from dictlearn [2], and understanding and implementing a sample implementation of scikit [4]. We adapted the latter to produce dictionaries for image gradient patches, to understand especially the looks of those, rather than what the rest of the material was intended for, image patches.

To better analyze the crooked dictionary learning process, we implemented a dynamic visualization of the dictionary while it is learning. This was done by writing a class inheriting from MiniBatchDictionaryLearning from scikit, to rewrite the dictionary learning method "fit" to plot the dictionary at every desired step inside the learning loop, while inheriting all other methods and properties. For customizable visualization of the dictionary while learning, some useful parameters were added, like in which step period should the dictionary be shown, a different step period at the beginning of the learning phase (because this is the most relevant phase for debuggin), show verbose information of convergence indications at every n-th step.

For the same goal, we also plotted the dictionary learning convergence over time. As we don't have access to the inner parameters of the actual learning algorithm, we plot the learning phase convergence as the difference between every two dictionaries of consecutive steps. This is based

on the observation that, for good dictionary learning runs, in the beginning the learned dictionary changes much, but in the end only details are tweaked by the learning algorithm.

The implementation of the distribution of the events into timeslices in the base implementation, creating a voxelgrid, was incorrect. This made the first timeslice stand out visually, and had the effect that the number of events in the voxelgrid didn't sum up (even when taking into account that positive and negative events at same pixel in the timeslice equalize each other). The error was that the linear voting in time neglected a big amount of the influence of the events at the end of the event stream. Instead of this, we used the linear voting algorithm from event_utils [7] to perform the event distribution into the voxelgrid.

To improve results, we trained on simulated data, using the esim simulator [6], and wrote scripts to format the simulated events from rosbags to our format of csv files, to only use specific parts of the simulated events and to choose a different edge velocity.

The *edge velocity* is defined as how many pixels a depicted edge moves in an event stream, in a certain direction. The dictionary learning algorithm is sensitive to this property of input data, because it incorporates edges and their motion over the course of the event stream part of the input.

An edge velocity of patch-size pixels per sequence (pix/seq), or below this, is advisable, because otherwise, the edge moves by the (simulated) camera so fast, that for one dictionary atom, not all corresponding timeslices can have information of the relevant edge.

2 Simulated Data

All event streams, real or simulated, are from DVS cameras, with a sensor size of 240 times 180 pixels. The main dataset used for simulating data for the experiments, was a grayscale image of the logo of the Berlinale. This produced better results than other input data. The simulator esim [6] was used to simulate event data of desired motion. With patches of size 7 x 7 pixels, we simulated data with edge velocity 4.5 pix/seq to 10.5 pix/seq and compared the results.

3 Outcome

When starting to analyze why our algorithm was not working, we had results as seen in Fig. 3. Even though the dictionary started to look like expected (Fig. 7 on the left) by tweaking edge velocity, patch-size, and others, the result looked nothing like we were expecting. The dictionary is divided in one row per atom, and from left to right one patch of the x derivative, one of the y derivative, five patches for positive events, and five for negative events, because the number-of-timeslices was set to five here. In this section, only experiment results with sufficient learning time and convergence are discussed.

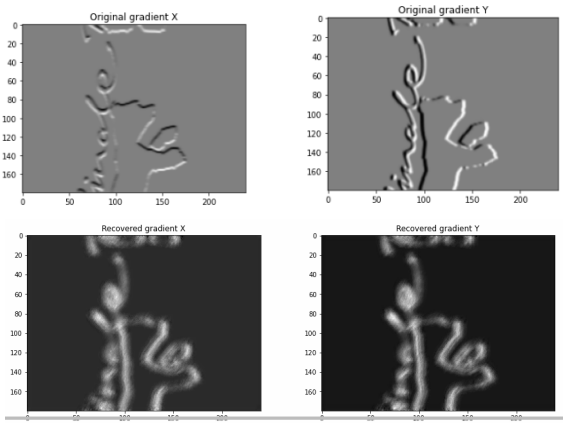

Figure 3: Recovered gradients for input Berlinale video with separate patches for positive and negative events

It was visible that only positive events were recovered. We realized over time, that these bad results were due to splitting up the event part of the dictionary in number-of-timeslices positive event patches, and number-of-timeslices negative event patches, instead of learning number-of-timeslices event patches for both positive and negative events together. In that way, less has to

be learned, resulting in a dictionary better modeling the event stream, with a given amount of learning data.

Analyzing the Berlinale video sequence, with the dictionary using unified event patches, we saw partially correct results for the first time. Still, these results had fuzzy mixes of white and black and, where the colors were more dominant, black and white were flipped, as can be seen in Fig. 4. The latter indicates that a positive gradient was recovered where there should have been a negative one and vice versa. Many of the atoms (rows) in the corresponding dictionary looked as expected as well, as can be seen in Fig. 7 in the middle. In this case, the dictionary is divided in one row per atom, and from left to right one patch of the x derivative, one of the y derivative, and finally five patches for positive and negative events together, because the number-of-timeslices is five in this case.

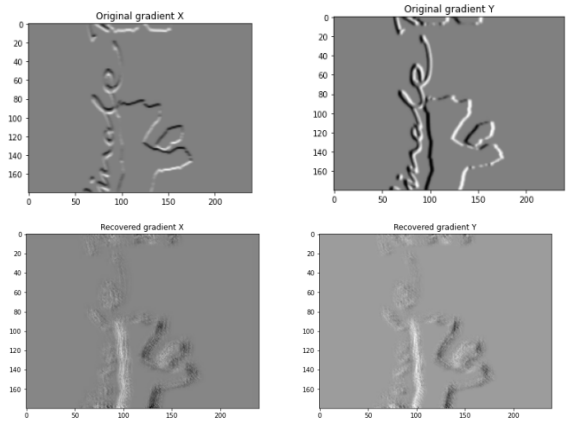

Figure 4: Recovered gradients for input Berlinale video with unified patches for positive and negative events

To enhance the recovering of the gradients, we switched to using simulated data, because with this, we are able to control the edge speed and edge direction. When working with simulated data, we realized that the flipped colors in the results (positive gradient where there should be negative and vice versa) can be influenced by the camera motion, because it triggers the events in the learning data. A simulated diagonal camera motion to the bottom right improves the results, in comparison to the other directions, which is probably attributable to the fact that with this camera motion, the image gradients and the event slices look very similar, which is easier to learn, than if those are dissimilar. Also, as discussed in Section 1.3, an edge velocity of patch-size pix/seq, or below this, is desired.

Applying these two concepts, we were able to recover the image gradients as shown in Fig. 5, with a patch-size of 7 x 7 and simulated data on the Berlinale image with an edge velocity of 5 pix/seq in x and y direction. It can still be seen that the lines are too broad, but the colors (i.e. positive / negative gradients) are definite and nearly all colors are at the correct location. The corresponding dictionary is shown in Fig. 7 on the right, and mostly has atoms which look as expected. Here, the dictionary is divided in one row per atom, and from left to right one patch of the x derivative, one of the y derivative, and then seven patches for positive and negative events together.

The choice of the edge velocity 5 pix/seq in x and y direction is justified also by the zoom into results shown in Fig. 6, where we see a velocity of 10.5 pix/seq in x and y direction on the left, of 5 pix/seq in the middle and of 4.5 pix/seq on the right. On the left, the reconstructed gradient in the hands of the bear is way more unexact than in the others, but on the right, the reconstructed gradient is not definite enough, e.g. at the back of the bear, compared to the other two.

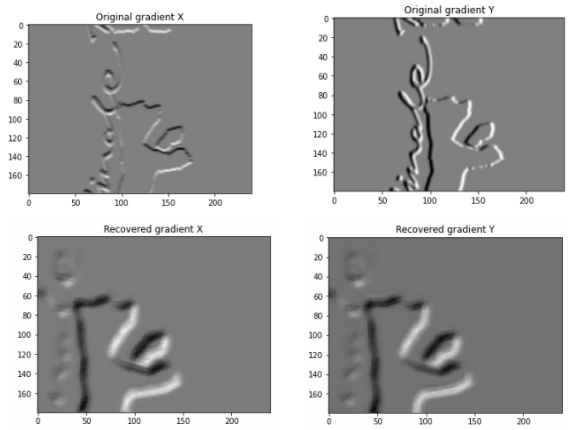

Figure 5: Recovered gradients for input simulated-Berlinale with unified patches for positive and negative events

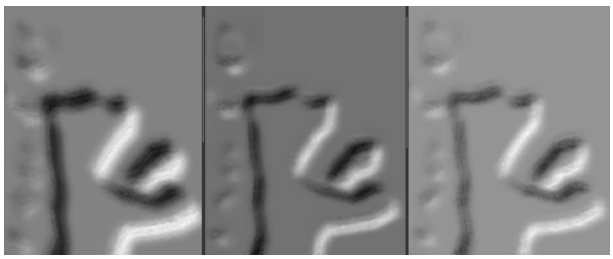

Figure 6: Zoom into the reconstructed y gradients for simulated-Berlinale, with on the left edge velocity of 10.5 pix/seq, in the middle 5 pix/seq and on the right 4.5 pix/seq.

References

[1] Souptik Barua, Yoshitaka Miyatani, and Ashok Veeraraghavan. Direct face detection and video reconstruction from event cameras. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, March 2016.

[2] Dictlearn developers. Dictionary learning tutorial. https://dictlearn.readthedocs.io/en/latest/tutorial.html. [Online; accessed 30-September-2022].

[3] Duke university Guillermo Sapiro. Digital image processing: p065 Introduction to Sparse Modeling Part [1, 2]. https://www.youtube.com/watch?v=h_fYIs0hdL0, https://www.youtube.com/watch?v=XLXSVLKZE7U, 2013. [Online; accessed 30-September-2022].

[4] Scikit learn developers. Image denoising using dictionary learning. https://scikit-learn.org/stable/auto_examples/decomposition/plot_image_denoising.html#sphx-glr-auto-examples-decomposition-plot-image-denoising-py. [Online; accessed 30-September-2022].

[5] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit. *CS Technion*, 40, 01 2008.

[6] Github users supitalp, christian rauch, and danielgehrig18. ESIM: an Open Event Camera Simulator, Github. https://github.com/uzh-rpg/rpg_esim, 2018. [Online; accessed 30-September-2022].

[7] Github users TimoStoff and tub rip. event_utils. https://github.com/tub-rip/event_utils. [Online; accessed 30-September-2022].

4 Appendix

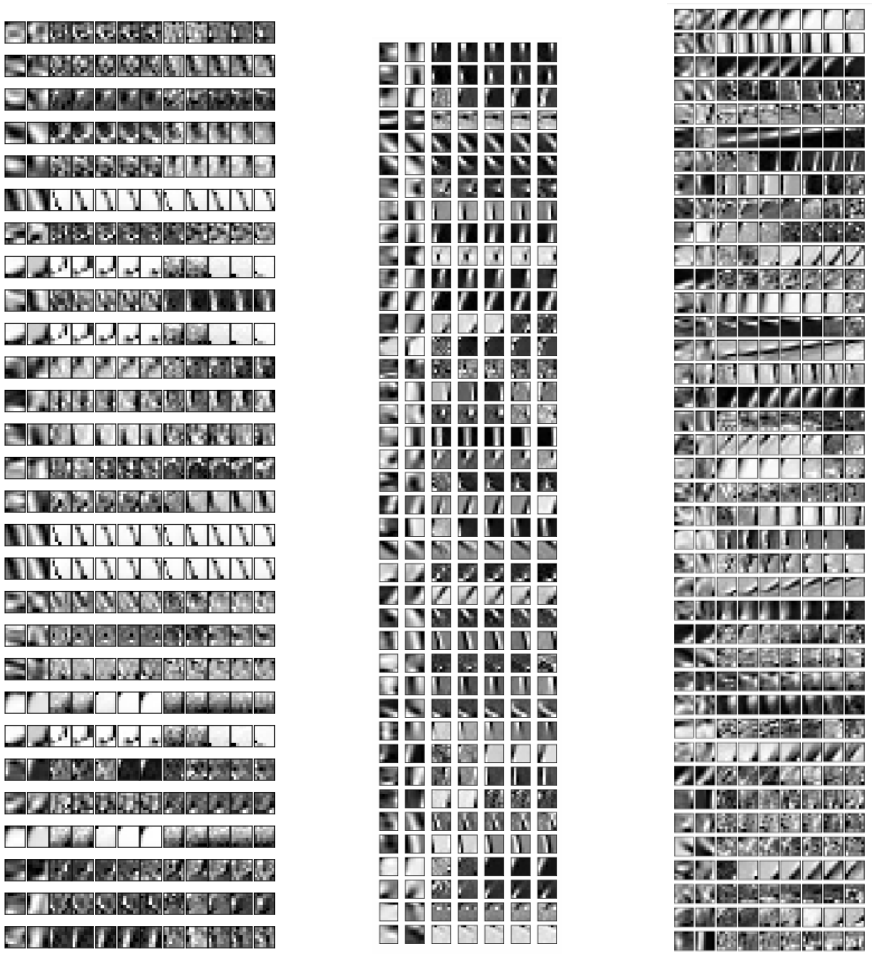

Figure 7: left: Dictionary for input Berlinale video with separate patches for positive and negative events. middle: Dictionary for input Berlinale video with unified patches for positive and negative events. right: Dictionary for input simulated-Berlinale with unified patches for positive and negative events.