# Image Reconstruction from DVS
# The constant battle agains MATLAB
# Student Project Report

Samuel Bryner
ETH Zurich
samuelbryner@student.ethz.ch

Marcel Geppert
ETH Zurich
mgeppert@student.ethz.ch

## Abstract

*// TODO - insert abstract*

## 1. Introduction

// TODO - insert introduction here

## 2. Related Work

Computer Vision using Dynamic Vision Sensors is a very young field since the sensors have only been around for a few years [5, 2]. While the problem of visual odometry and visual SLAM has been known for many years and many approaches have been proposed, the work by Kim *et al*. [4], which is the basis of our work, is, to our best knowledge, the first approach to the problem using a DVS. Other research with DVS's focuses in great parts on object tracking [7, 3] or pose tracking [6], respectively.

## 3. Dynamic Vision Sensors

A Dynamic Vision Sensor (DVS) [5], also known as an event camera, is a new kind of camera. In contrast to a standard camera it does not output a complete image of the scene, but a series of events. An event is generated when the sensed brightness of a pixel changes more than a certain threshold. This happens for each pixel completely independent from all the others.

A DVS has several advantages compared to standard cameras. Since there is no need to gather data from all pixels to generate an image, events can be sent with an extremely low latency of 15 $\mu$s or less [5, 2]. For the same reason, there is practically no motion blur in the DVS signal. Another benefit of independent pixels is the very high dynamic range. Saturation, blooming or smearing as observed when a bright light source is captured with a standard camera is basically nonexistent. Finally, due to the reduction of the camera signal to changes and the resulting elimination of redundant information, the signal bandwith is much smaller than with a standard camera.

All these features make the DVS a very desirable sensor for motion tracking, especially for mobile robots where the data has to be analyzed on constrained hardware.

However, there are currently several caveats: First of all, currently available DVS (DVS128 [5], DAVIS [2]) have a very limited resolution of 128 $\times$ 128 or 240 $\times$ 180 pixels, respectively. This obviously limits the spacial accuracy of the sensed data, but it is likely to be a simple matter of time until sensors with a higher resolution are developed. A more important point is that the computer vision algorithms that have been developed during the last decades can either not be used at all or have to be adapted to the new data representation.

## 4. Our Work

// TODO - insert section to explain what we did here // TODO - move core algorithm section before this?

-implemented paper (kim) in matlab
-some simplifications
-use particle filter directly with euler angles (ignore corner cases)
-start with first fov in map

### 4.1. Camera Simulation

In order to facilitate development we implmemented a full DVS simulator that generates events from a spherical panorama. This allows to develop and test each part of the algorithm independently with perfect input data and ground truth. It also makes experimentation easier as everything can be inspected at will.

The simulation starts off with a normal image containing a 360°panorama. The field of view of the camera is computed given its orientation (normal Euler angles) and the panorama image (the ground truth map) is sampled us-

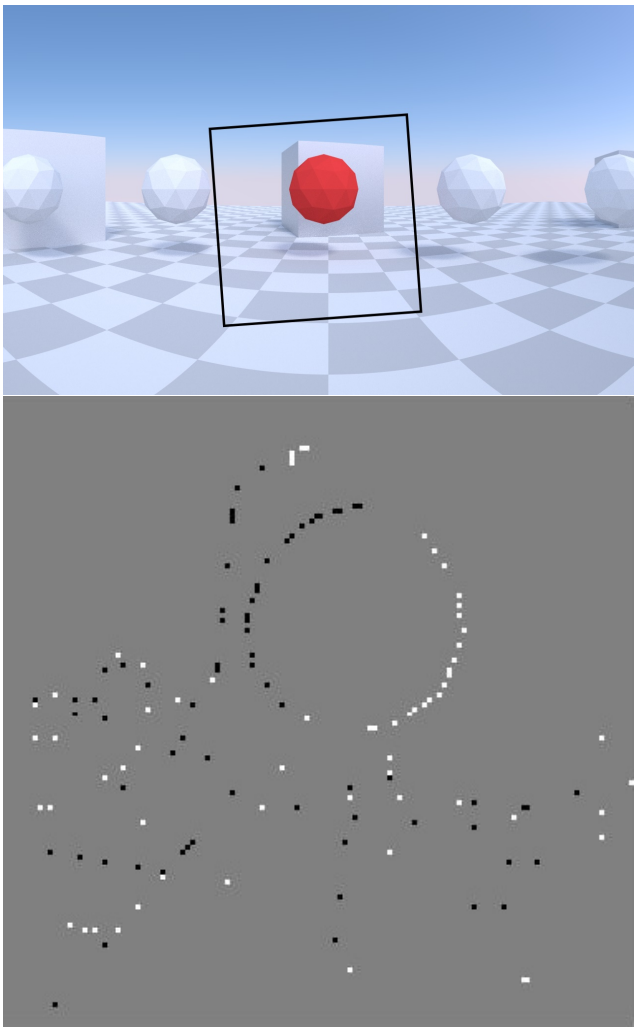

Figure 1. above: the map used as input with the camera's FOV, below: events generated by moving the camera to the right

ing raycasting. The camera is then rotated by a tiny amount to get a second image patch. These patches are then subtracted to get a list of events where the difference exceeds a threshold.

This discrete approach is fairly straightforward but suffers from the problem that it is very slow, as very tiny steps have to be taken to get good events.

# 5. Core Algorithm

Similar to most SLAM algorithms, our algorithm consists of two co-dependent steps. In the first step we try to estimate the current orientation based on the camera signal, our map of the environment and the previous orientation. This is then used to write the new data to the map and reconstruct a grayscale image. This iteration is performed every time an event arrives from the camera. Both the tracking and the rotation work on the assumption that the output of the other one is correct.

We assume a static scene, so every event (every change in brightness) is caused by camera rotation or is due to noise.

Further, we only track camera rotation; We assume there is no translation at all and therefore no parallax displacment either. This greatyl simplifies our algorithm as we only have to keep track of a two dimensional map.

To speed up things even further, we assume that the camera's initial field of view is already known and included in the map. With this assumption we do not have to deal with a special initialization procedure and can immediately start with the standard iteration while being relatively sure that the camera movement is tracked correctly.

This is not an unreasonable assumption, as never DVS models incorporate the ability to take full-frame pictures and a simple method for generating full images can be used with any model (see Section about Experiments [TODO: reference]).

## 5.1. Rotation Tracking

In order to integrate intensity change events into a full gradient map of the environment, we must track the current position and movement direction of the camera.

The camera's position is represented using a particle filter, where each particle consists only of a weight and the three Euler angles necessary to describe the camera's orientation.

Whenever a new event is received, we disturb the particles randomly with variance proportional to the time since the last event. This is essentially a constant position model, where the camera is assumed to stay stationary between events but with uncertainty growing with time.

To update the weights of the disturbed particles, we retrieve the position of the camera at the time of the last event of the *same* pixel (which can be a lot earlier than the previous event). We then sample our map (from the reconstruction part and which we assume is essentially correct) at this earlier position. This intensity is then compared with the intensities at all the proposed current positions: The closer the intensity difference to the intensity threshold that generates an event, the liklier is the new position and the more weight this particle gets.

Finally, the particles are resampled whenever the effective number of particles (one over sum of squared weights [TODO: insert formula here?]) falls below some threshold ([4] uses N/2). Resampling is done by copying particles with probability equal to their weight into a new set.

### 5.1.1 on the matter of prior positions and accuracy

A particle filter is a nice way of keeping track of multiple hypotheses as might conceivably occur when updating on a single event: An event might match multiple edges and

we might only be able to resolve the correct position after receiving further events.

But as described above, we need to know the position at the time of the previous event of the same pixel. And as the pixels are independent, we would have to keep a full copy of the particle filter for every pixel. This not only requires a lot of memory, it also considerably slows down measurement updates as we now have to compare each of the current particles with all possible previous ones.

However, our experiments show that it is actually enough to only keep the weighted average over the particles as any errors are quickly averaged out by the sheer number of events.

### 5.2. Scene Reconstruction

We use Kalman Filters to reconstruct a gradient map from the noisy DVS signal. These gradients are then used to generate a grayscale image by applying a poisson reconstruction algorithm [1]. For each pixel in the output image there is a Kalman Filter that keeps track of the color gradient at the pixel's position. The input to the Kalman Filter is the event frequency on along the current movement. Here we assumed that the pixel's movement was constant since the last event it triggered. While this is obviously not true all the time, it is a good approximation due to the high rate of events. Note that the pixel movement is not necessarily parallel to the camera movement, e.g. if the camera is rotating around its z-axis.

The pixel movement is computed with the same formulas as used in the simulation and the map lookup during tracking. Given a camera orientation, the orientation of the ray through the pixel and the camera's focal point is computed. This is then mapped to a pixel in the output image. While we use the exact position in the output image to compute the pixel movement, we do not, however, interpolate between several pixels when writing the signal to the map, but simply round the position in the output image to the closest pixel. Since we do not write the signal directly but use Kalman Filters to reduce the noise, this simplification greatly reduces the complexity of the algorithm compared to dividing the signals between several filters.

The exact formulas used for the Kalman Filters can be seen in [4].

## 6. Problems

// TODO - insert section about problems

-map initialization

-runtime

→ couldn't test with camera

-MATLAB

Figure 2. image of our lab generated by integrating over short interval while removing shutter

## 7. Experiments

Most of our experiments are based on the simulation described in [TODO: reference to simulation section] as our code is not perfomant enough to handle the amount of events a real dynamic vision sensor generates and because the simulation makes comparisons with ground truth much easier.

There are a few things we looked into with the real DVS tough:

### 7.1. initialization by removing shutter

If the DVS is fully covered by a shutter we can recover a full picture by simply removing it slowly and integrating all the events generated. This allows for an easy initialization of the map using only the DVS. An example is shown in Figure 2

### 7.2. camera calibration

As with any sensor, we must calibrate the DVS to get usable data. For normal cameras, this is usally done by capturing images of a checkerboard pattern from different viewpoints and subsequently rectifying images so they fit into a standard pinhole camera model.

Unfortunately, the dynamic vision sensor only detects changes and cannot directly capture a calibration pattern. We could take pictures as described in the previous section, but for calibration there is an even better method [TODO: scite Scaramuzza paper]: Use a computer screen to quickly show and hide the pattern. This is not only quicker and

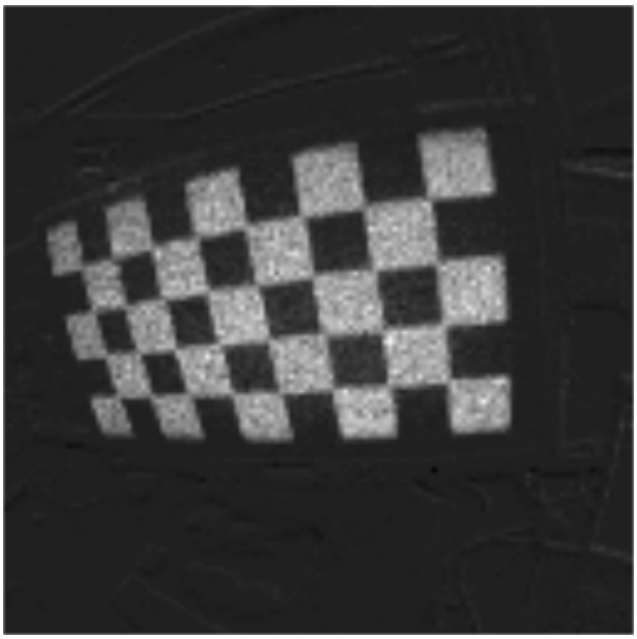

Figure 3. a sample checkerboard pattern used for calibration

more convenient than removing a shutter, it also has the advantage of only showing the pattern and none of the surrounding scenery.

This method can also be used to focus the camera optics by displaying a pattern of lines with varying sizes.

## 8. Results

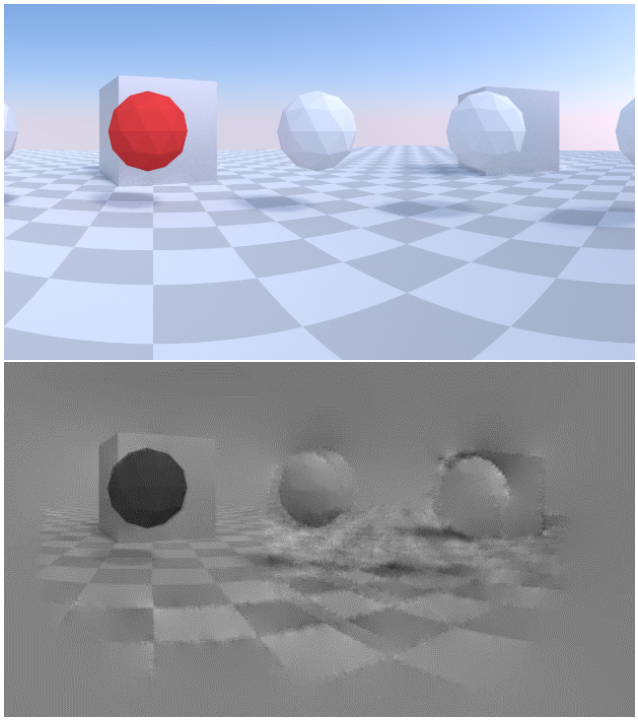

Figure 4. result of full algorithm running in simulation

## References


[1] Matlab code for poisson image reconstruction from image gradients. http://web.media.mit.edu/~raskar/photo/code.pdf, 2015. [Online; accessed 07-March-2015]. 3

[2] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck. A 240 × 180 130 db 3 $\mu$s latency global shutter spatiotemporal vision sensor. *Solid-State Circuits, IEEE Journal of*, 49(10):2333–2341, Oct 2014. 1

[3] J. Conradt, R. Berner, M. Cook, and T. Delbruck. An embedded aer dynamic vision sensor for low-latency pole balancing. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 780–785. IEEE, 2009. 1

[4] H. Kim, A. Handa, R. Benosman, S.-H. Ieng, and A. J. Davison. Simultaneous mosaicing and tracking with an event camera. In *British Machine Vision Conference (BMVC)*. BMVC 2014, 2014. 1, 2, 3

[5] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128 × 128 120 db 15 $\mu$s latency asynchronous temporal contrast vision sensor. *Solid-State Circuits, IEEE Journal of*, 43(2):566–576, Feb 2008. 1

[6] E. Mueggler, B. Huber, and D. Scaramuzza. Event-based, 6-dof pose tracking for high-speed maneuvers. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2761–2768. IEEE, 2014. 1

[7] D. Saner, O. Wang, S. Heinzle, Y. Pritch, A. Smolic, A. Sorkine-Hornung, and M. Gross. High-Speed Object Tracking Using an Asynchronous Temporal Contrast Sensor. In J. Bender, A. Kuijper, T. von Landesberger, H. Theisel, and P. Urban, editors, *Vision, Modeling & Visualization*. The Eurographics Association, 2014. 1