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

// TODO - insert related work here
-Kim *et al.*
-Scaramuzza
-[4]
-vSLAM?

3. Dynamic Vision Sensors

A Dynamic Vision Sensor (DVS) [4], 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 μs or less [4, 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 [4], DAVIS [2]) have a very limited resolution of 128×128 or 240×180 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
-implemented paper (kim) in matlab
-some simplifications
-use particle filter directly with euler angles (ignore corner cases)
-start with first fov in map

5. Core Algorithm

Similar to most SLAM algorithms, an iteration of our algorithm consists of two 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.

5.1. Rotation Tracking

// TODO - insert section about tracking
Assumptions:

- event (change in brightness) is caused by camera rotation or noise, but not by movement in the scene ($\rightarrow$ static scene)
- only rotation, no translation and therefore no parallax displacement
- initial FOV is already known and in the map

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 [3].

In the current state of our code, 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]).

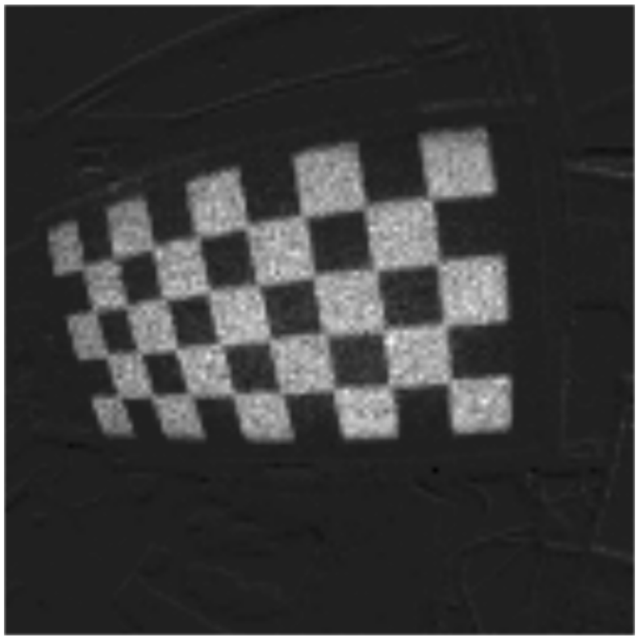
6. Problems

// TODO - insert section about problems
-map initialization
-runtime
$\rightarrow$ couldn't test with camera
-MATLAB

7. Experiments

// TODO - insert section about experiments

- initialization
  - integrate while removing shutter
  - integrate over short interval ($\rightarrow$ outlines)
- camera calibration



7.1. Camera Simulation

In order to facilitate development we implmemented a full DVS simulator that generates events from a spherical panorama. This allows us 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 using 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.

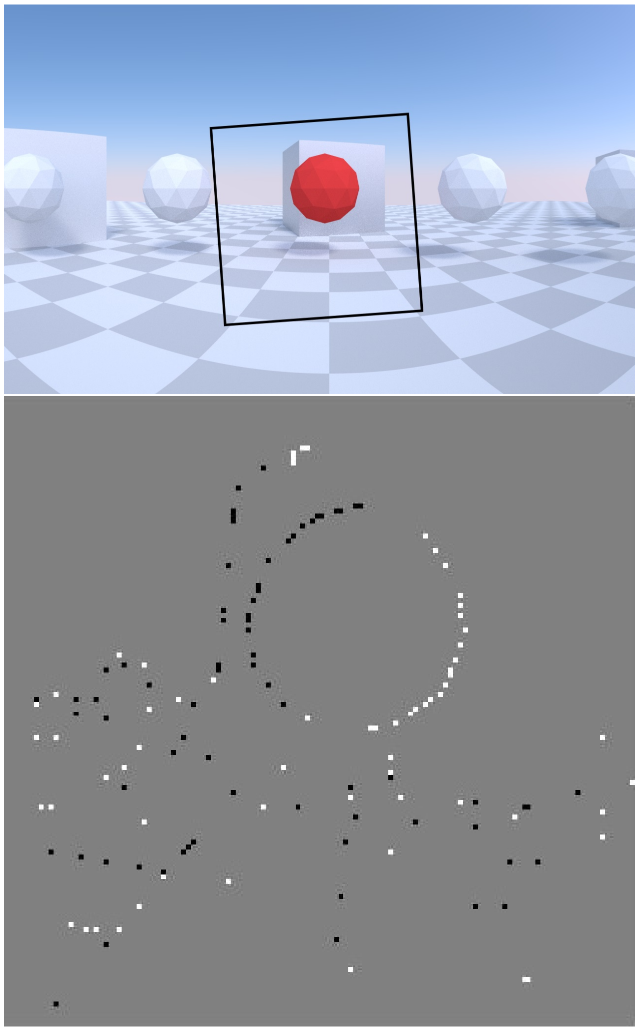


Figure 1. above: the map used as input with the camera's FOV, below: events generated by moving the camera to the right

8. Results

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].

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

[3] 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.

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

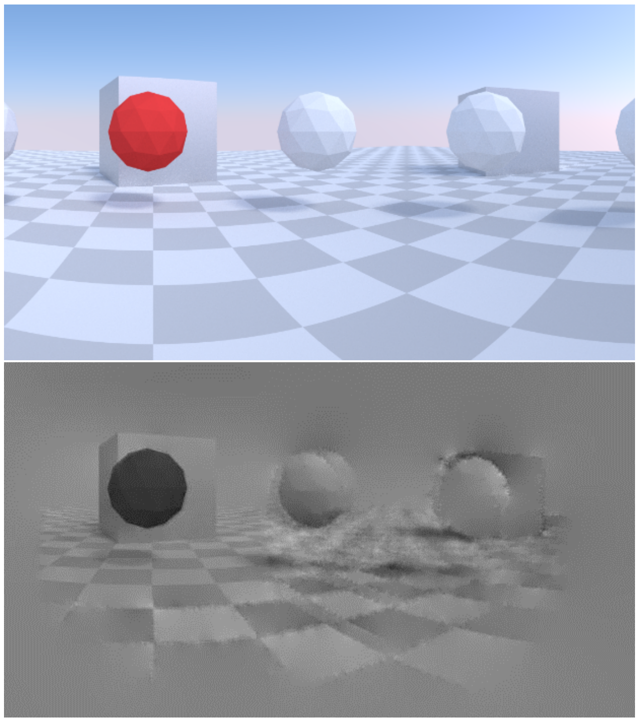


Figure 2. // TODO

Figure 3. // TODO