Motion Estimation from Sparse Time Correlated Image Data

Andrew Mueller
California Institute of Technology
1200 E California Blvd, Pasadena, CA 91125
amueller@caltech.edu

Abstract

Novel image sensors produce a stream of electrical pulses corresponding to photon detections at pixel locations rather than image frames at discrete times. Image processing techniques like object detection and tracking are classically applied to images or frames of video. While images can be produced by these sensors through integration of photon counts over the sensor, this process throws away the high timing resolution of photon detections present in the original data-stream. Therefore I propose a method for motion estimation directly from a highly time correlated steam of photon detections. This technique can enable motion estimation and tracking with these novel image sensors.

1. Introduction and Motivation

User friendly and high-resolution image sensors made from arrays of single photon detectors are now under development. The leading photodetector technologies that make up these sensors are Single Photon Avalanche Photodiodes (SPADs) for visible to near-infrared sensitivity, and Superconducting Nanowire Single Photon Detectors (SNSPDs) for mid-IR sensitivity. These sensors generate a series of asynchronous electrical pulses corresponding to the arrival of photons at individual pixels. These pulses may be integrated to form an image, but one of the most valuable features of these arrays is the high timing resolution they provide for each photon detection event. SNSPD arrays can record the arrival time of photons to within hundreds of picoseconds, the time it takes a photon to travel a few centimeters. If the photon detection events are integrated into frames, this timing information is discarded or strongly subsampled.

It is possible that useful motion information can be gained from the asynchronous photon detection stream on timescales much shorter than that needed to form one or two high quality frames. This is important because many SNSPD and SPAD arrays have multiplexing imposed maximum count rate restrictions that keep them from integrating photons into high quality frames as fast as high speed CMOS based cameras [4]. A neural network may be able to make up for this limitation in the cases where the imaging system is used to estimate motion or track objects. With this limitation circumvented, a SPAD or SNSPD array imager brings with it all the benefits of time-resolved single photon counting including ultra-high sensitivity and potential integration with LIDAR.

A group at Caltech and JPL is investigating how an SNSPD array might be used as an imaging system for vehicle navigation in dark environments where only the infrared thermal emissions of objects can be captured and used for triangulation and control.

Other publications have investigating the use of neural networks to optimize the utility of an asynchronous photon detection data stream from novel image sensors [2], [1]. However these works either used the data to generate higher quality images instead of motion information, or to detect objects in a stationary scene.

2. Process Overview

2.1. Prediction of Flow Fields

For this project the objective is made more specific by choosing that the output of the neural network be a so-called flow field. A flow field is an array of 2d vectors, one for each pixel of the imaging array. The vectors point in the direction of the motion of the surface patch observed at each pixel. Flow fields are often used with optical flow algorithms to interpolate motion between frames or track objects through video. If a flow field can be generated from raw image data, it is very useful for any subsequent object tracking or 3d parallax analysis.

In order to train a machine learning algorithm to estimate motion in this way, a large dataset with sparse time-correlated photon counts and correlated ground truth flow fields was needed. As labeling individual pixels of real-world video with motion vectors is not practically feasible for real world video, training datasets for learned optical flow estimation are produced with 3d graphics. Com-

puter simulations of moving objects and cameras may be rendered along with a corresponding flow field render pass.

2.2. Time discretization into Sub-Frames

The proper data format to feed into the neural network is a sequence of time-tagged photon detections corresponding to pixels on an image plane. Ideally a detection could come from any time uniformly sampled from some time window. However due to the limits of the 3d graphics data creation process, the uniform-time datascructure had to be approximated by discretizing it into a series of shorter times on which a so-called sub-frame of animation is rendered. This way a single training example is made from a series of sub-frames from which detections are sampled, with each detection gaining the timing information of the sub-frame it originated from.

Ideally at least 256 sub-frames would have been rendered per example to provide accurate timing information for each photon detection. But render time of the datasets scaled with sub-frame number such that rendering large sets with just 64 or 128 sub-frames already took more than 30 hours.

2.3. Scene Generation and Motion Statistics

Thanks to the extensive python API of the 3d graphics program Blender, setup and rendering of datasets could be largely automated while still ensuring that each training example created was sufficiently unique.

The moving object was added to the scene and animated in such a way that it's rotation, translational velocity, and rotational velocity was chosen randomly from uniform distributions with fixed bounds. To animate the object's velocity, it's starting location was first set to a random point on a sphere of radius 1. This was done by randomly generating the angular coordinates of a point on the sphere and converting that to a Cartesian coordinate. This location and randomised Euler angles defining the object's orientation were key-framed at $t = -1$ second. At the end of the animation, t = 1 second, the object was key framed at the opposite point on the sphere, with another set of randomized Euler angles. This way it moved at a constant speed through the origin at $t = 0$ and with rotational velocities around any axis between 0 and π radians per second. For all examples the camera was stationary and pointed down along the z-axis at the scene origin.

Textures were also applied to the object in a randomized way. For each example, one out of 7 image textures was chosen and applied. The set of 7 textures used consisted of pictures of common materials like wood, rock, sand, and rusty metal. The texture mapping coordinates were also randomized with appropriate scaling factors such that no two objects with the same texture looked exactly the same.

Scene lighting was also partially randomized. A low intensity environmental light was present in every scene so that there were no harsh shadows. Three 'sun lamps' were also added that projected light on the moving object from random directions.

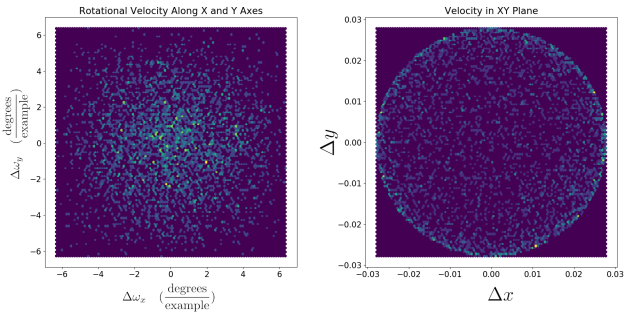

Figure 1. Histograms of Δx with Δy, and $\Delta\omega_x$ with $\Delta\omega_y$. Other combinations like a histogram of Δy with Δz or $\Delta\omega_y$ with $\Delta\omega_z$ look comparatively similar.

Figure 1 shows the distribution of velocities in the xy plane and rotational velocities along the x and y axes of the 2nd dataset (The portion of the script that determined these motion statistics was the same for the 1st dataset, though the statistics were not recorded for that set). These histograms suggest that there are no major unintended correlations between rotations and positions, or between rotational velocities and orientation of the moving object at the time it passes in front of the camera. The network is therefore not able to directly infer the rotation or velocity of an object simply from the shape of its silhouette, which in the first dataset is clearly identifiable.

2.4. Object Choice

The 3d mesh used as the moving object in all examples of both datasets is the unofficial mascot of the 3d graphics software Blender, a stylized monkey head named 'Suzanne'. This is a standard object in the program and is an alternative to other standard objects like the Utah teapot or Stanford Bunny that make their way into many 3d graphics related publications. It was chosen because a simple object with some nontrivial shape was needed so that each example rendered quickly yet looked sufficiently dissimilar from any other examples in the set.

The single object was chosen to keep the dataset uniform in some respects and allow for easy learning. However, this mostly likely keeps the network from being as general as it could be. If the network developed here was to be used on image data from a real-life scene, it would have to be trained on a dataset more representative of real life or the specific analysis objective at hand. [1]

[1] Unfortunately a new test set showing a moving object of a different shape could not be rendered and processed before the deadline. This is

2.5. Sampling from Sub-frames

The RGB subframe sequence was first converted to luminosity. The luminosity of each pixel in each subframe was then used by multiplying it by a scaling factor and setting the product to be the mean of a Poisson distribution from which so-called 'photon detections' could be drawn. This was set so that the mean photon detection probability over each pixel of each subframe when sampling from a 100% white subframe pixel was 0.0025. This resulted in about 60 detections generated per subframe.

The scaling of the Poisson distribution was chosen by considering two factors. First, the mean photon rate per pixel needed to be large enough so that a large fraction of the image area received at least one detection over one example. The rate also could not be so large that pixels often received much than one detection per example. This is because the data structure chosen for input into the network discarded all information for detections following the first one for each pixel. While this is a source of lost information that does not make it to the network, it shares similarities with the properties of the true image sensor systems I am attempting to simulate. Both SPAD and SNSPD arrays have relatively long reset times for pixels in the array following detections, ranging from 10 nanoseconds to multiple microseconds. During this reset process the pixels are not photosensitive. If the motion studied with one of these image systems is high speed (e.g. the sensor is used for an application normally requiring a very high-speed camera), it is possible that appreciable object displacements could occur on the timescale of the reset process.

The datastructure input into the network is a 3d tensor with rows and columns of an image, and 2 channels for each pixel. The first channel is a Boolean that is set high if a detection was recorded at that pixel location during the example, and the second channel is the time of detection, determined from the subframe number from which it was drawn. The Boolean channel was included in the datastructure so that the network would be able to discriminate between pixels that did not record a detection for the entire example, and detections that occurred near t=0, when the value in channel 2 is near zero.

2.6. Network

The network architecture chosen for this motion estimation task is based off of the original FlowNet publication by Dosovitskiy et al. [3]. This paper, which demonstrated flow field generation from sets of video frames, presented two network architectures, one of which was more customized to the use of discrete video frames as input. The other network architecture presented, FlowNetSimple, was more general and amenable to a novel data-structure as input. It shares much in common with other computer vision deep network architectures like those used for object localization and semantic segmentation. It is made exclusively with convolutional layers and has a contractive part and an expansive part. For use in this project, the network was first scaled down along all dimensions by a factor of two. This was done to decrease training time and allow for faster iterative improvements. Further changes to the network, optimizer, and loss function were made to improve prediction performance for two datasets, as described below.

The loss function of this network is the End Point Error (EPE). It is the squared norm of the difference between motion vectors of predicted and ground truth flow fields. It is calculated for each flow field pixel and averaged over the image area.

3. Dataset 1

6051 sequences of sub-frames were rendered with Blender showing an object moving through the center of the frame with randomized rotation and velocity as described above. There was no background present in this dataset, and the number of sub-frames was set to 64 for this example. These sub-frames were using the Poisson distribution methods described above, and the dataset was split into train and test sets, with the train set consisting of 75% of the examples. The dataset was then used to train the downsized FlowNetSimple network. Figure 2 shows the contractive part of the network architecture and an example of the input layers. Initially, the end point error diverged with training steps despite a series of batch normalization steps at various stages of the network. This was avoided by replacing the default ResNet50 normalization parameters of the dataset with custom computed average and variances of the two channels. With the properly normalized dataset and a tuned Adam optimizer, the network was able to improve end point error with each minibatch. Other optimizers were tried like Adadelta and SGD, however they either made endpoint error diverge with each minibatch, despite the properly normalized dataset, or simply never reached as low EPE values as those achieved with Adam.

With the tuned Adam optimizer and the downsized FlowNetSimple architecture, EPE converged to around half of it's untrained value. However, the predicted flow fields showed significant inaccuracies. The flow direction of object edges appeared well predicted, but the interior area of the moving object would appear blotchy with various inaccurate directions predicted in these areas. The issue was remedied by adjusting parameters of the multiscale loss function used by this network. The loss function penalizes not just EPE on the final flow field generated at the end, but also on lower resolution flow fields produced at earlier stages in the expansive part of the network. How much the

due to the added complexity of working with a non-standard object in the generation script, and time it would have taken to render enough new test examples.

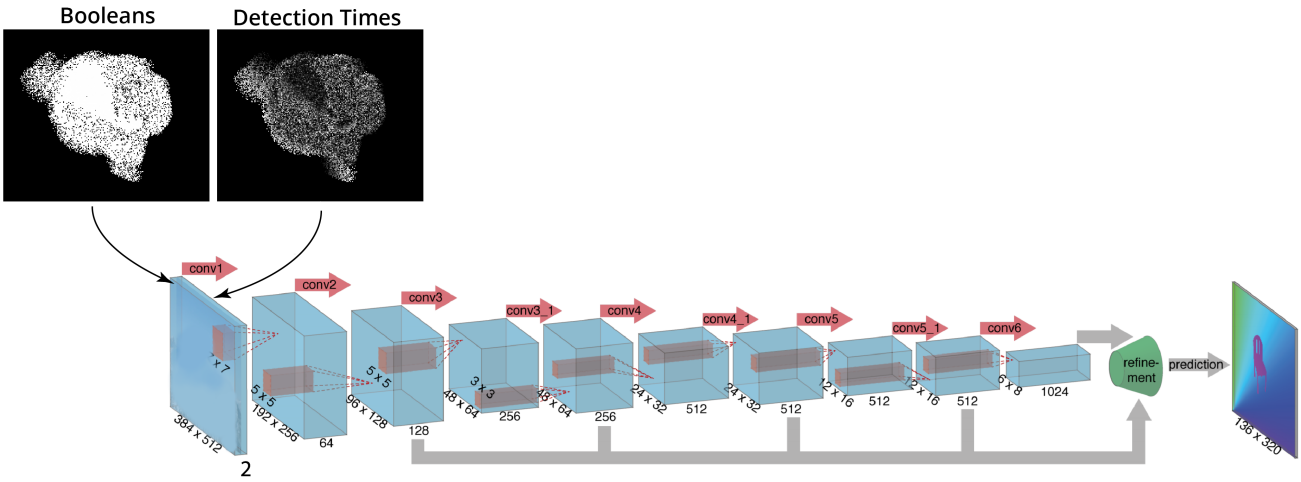

Figure 2. FlowNetSimple network with time correlated photon detection input. Modified image, with original network from [3]

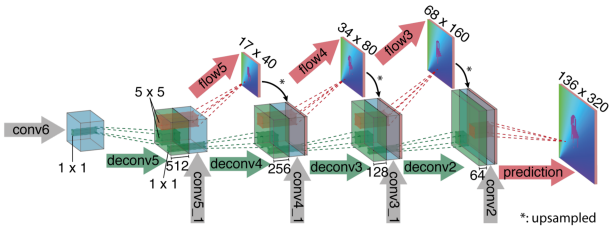

Figure 3. Expansive part of the network. The weighting of the EPE computed for the flow fields produced by flow5, flow4, and flow3 can be adjusted. Image from [3]

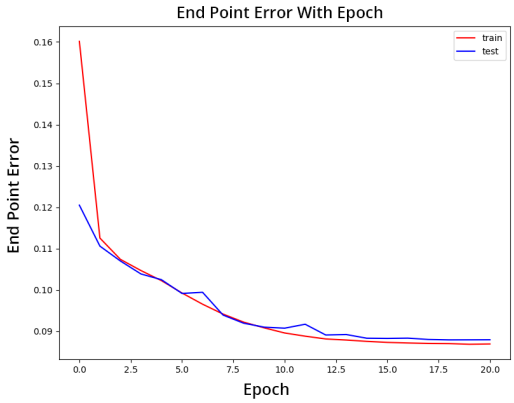

Figure 4. End point error on train and test set for Dataset 1 with Epoch. From the training iteration that generated the 100% image examples shown in figure 5b)

total loss depends on the end point errors from these small flow fields can be tuned. I increased the weighting of EPE computed for the smaller flow fields with the perception that this would force the network to put more emphasis on properly estimating the motion of the overall image or large sections of it instead of focusing on detail areas. This significantly improved the predictions (20% drop in EPE) on the test set. The thickness of the first two convolutional layers was also doubled, and this improved results slightly (5% drop) as well. Convergence of EPE with epoch is shown in figure 4. I found increasing the size of other layers in the network did not improve results, while increasing the sizes of layers near the center only improved results on the train set (over-learning). Figure 5 a) shows how EPE scaled with the size of the training set, and b) shows the degradation of flow field quality with the smaller training set.

The network still had some trouble estimating the motion of interior patches of the moving object in situations where those patches were not near to a boundary. This suggests the network still learned most about the object's motion from the motion of its boundary or silhouette, which was easy to identify even with just a few detections in an example.

4. Dataset 2

A network that can estimate motion not just by identifying and tracking the boundary of objects but also their surface textures is desirable. The design of the 2nd dataset was tailored to determine if the FLowNetSimple architecture could learn this ability.

Compared to dataset 1, the process of rendering subframes for the 2nd dataset was different in 2 ways. First the number of sub-frames was increased from 64 to 128, thereby doubling the temporal resolution of detections in each example. Second, a background was added with its own randomized velocity and surface texture.

In order to inform the network about object motion from contrasts in object texture, the down sampling process on the sub-frames was modified to output 3 color channels instead of counts drawn simply from the luminosity chan-

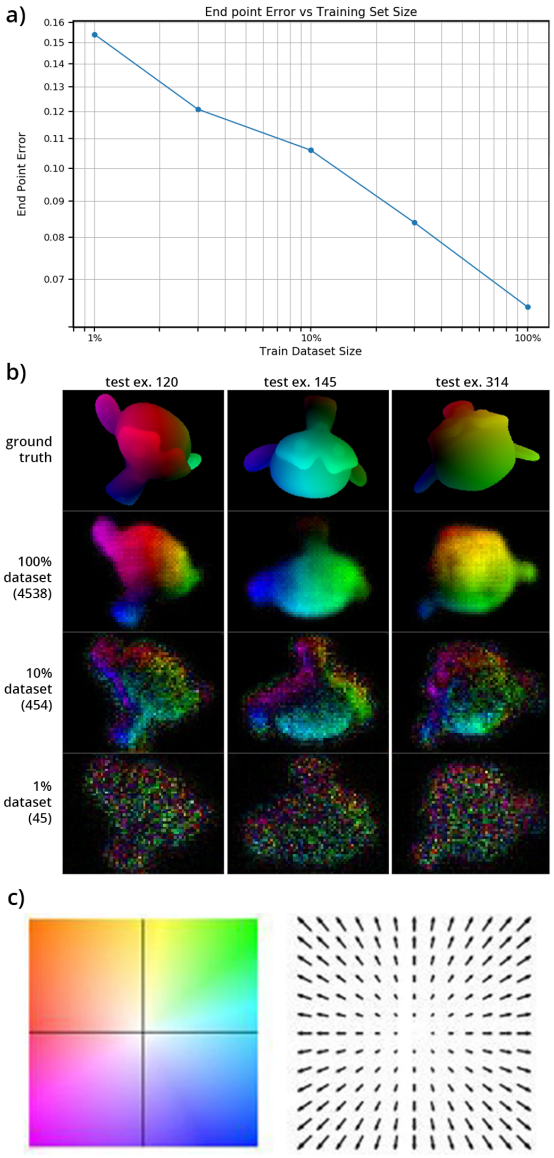

Figure 5. a) Converged end point error with respect to training set size. The end point errors plotted were found by training for the number of epochs necessary so that subsequent epochs did not lower end point error by more than 1%. b)While decreasing the size of the training set by 100x does not increase the end point error drastically, the degradation in motion prediction is clear from predictions on examples of the training set. c) Color legend for interpreting flow fields. The color of a pixel identifies the direction and magnitude of the vector at that pixel (slight inaccuracy: the legend shows gray as corresponding to a zero magnitude vector. The flow fields shown here map zero magnitude to black instead.

nel of sub-frames as was done for the first dataset. This change is inspired by certain multispectral image systems

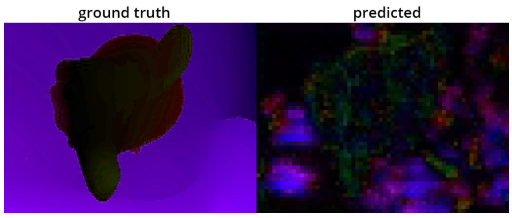

Figure 6. Ground truth and predicted flow field for an example from Dataset 2 test set.

that are under development at JPL and Caltech, and that use the kind of single photon detector arrays that inspire this project. The sub-frame rendering and down sampling process used here is not a precise mathematical model of an specific imaging system of this type, but it has general features in common with real world systems.[2]

This dataset was also split into a train and test set and property normalized. The tensor input into the network had 6 channels for each pixel, corresponding to 3 sets of Booleans and detection times for the 3 color channels.

Despite further adjustments to the optimizer and multiscale loss function, and changes to convolutional layers sizes early in the network, the predictions generated on the test set for the second dataset were poor. As shown in figure 6 the network after sufficient training time was able to gather some information about motion of the background and foreground object but only in some isolated areas that presumably had high contrast color or luminosity features. Many of the flow fields predicted from the test set also had good predictions on the motions of object edges, but poor predictions on large areas of background or on the interior of the foreground object. Other attempts to improve these results included increasing the kernel sizes used in the first few convolutional layers. This seemed to improve the end point error overall, but at the cost of clarity of the flow field. The larger kernels essentially blurred the flow field to the point where the foreground object was not recognizable.

It appears this multi-color dataset is crippled by the requirement that there be only one recorded photon detection per pixel per color channel. This was probably insufficient to allow tracking of texture features through time.

5. Conclusion

As the development of a network for predicting motion from these sorts of image sensors may prove useful at later stages of my PhD research, I hope to continue working on this project by generating a hyper-spectral dataset with more

[2]Experimental SNSPD imaging systems under development have much higher spectral resolution than the 3 channels used here, from 100 to 1000 channels across several hundred nanometers in the infrared. A dataset with many more spectral channels is desirable, but was not produced here due to the complexity involved with simulating hyper-spectral textures.

realistic attributes. I also hope to base the down sampling process on a more accurate mathematical model of a real SNSPD array, taking into account other performance tradeoffs inherent in the device beyond the post-detection dead time discussed above.

I hope that applying a neural network like the one used here to an accurate mathematical model for a SNSPD array camera will show it can gather enough information about the motion of objects in view from color or luminosity contrasts on surfaces. Showing this is possible expands the potential use cases of these sorts of imaging systems, especially during the next few years of development when their maximum count rate limitations make them an uncertain choice for motion tracking and control applications.

6. Acknowledgements

- Clement Pinard `https://github.com/ClementPinard` for the PyTorch implementation of the FlowNet publication

- Tobias Weis `http://www.tobias-weis.de` for functions and techniques for converting .exr files from blender with vector passes into numpy array flow fields.

7. Code and Video

The code used for this project is available here: `https://github.com/sansseriff/ee148_FinalProject`

A video showing the unaltered sub-frames, sampled sub-frames and ground truth flow fields is available here: `https://youtu.be/bQuncWoNS2Y`

References

[1] P. Chandramouli, S. Burri, C. Bruschini, E. Charbon, and a. A. Kolb. A bit too much? high speed imaging from sparse photon counts. pages 1–9, 2019.

[2] B. Chen and P. Perona. Vision without the image. *Sensors (Basel, Switzerland)*, 16(4):484, Apr 2016. 27058543[pmid].

[3] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. pages 2758–2766, 2015.

[4] A. D. B. R. M. B. B. A. K. B. B. M. D. S. Jason P. Allmaras, Emma E. Wollman. Demonstration of a thermally-coupled row-column snspd imaging array. *arXiv:2002.10613*, 2020.