Event-based 3D Motion Detection with Depth-Dependent PSFs

Adam Cahall
Cornell University
Ithaca, New York, United States

Haley Lee
Cornell University
Ithaca, New York, United States

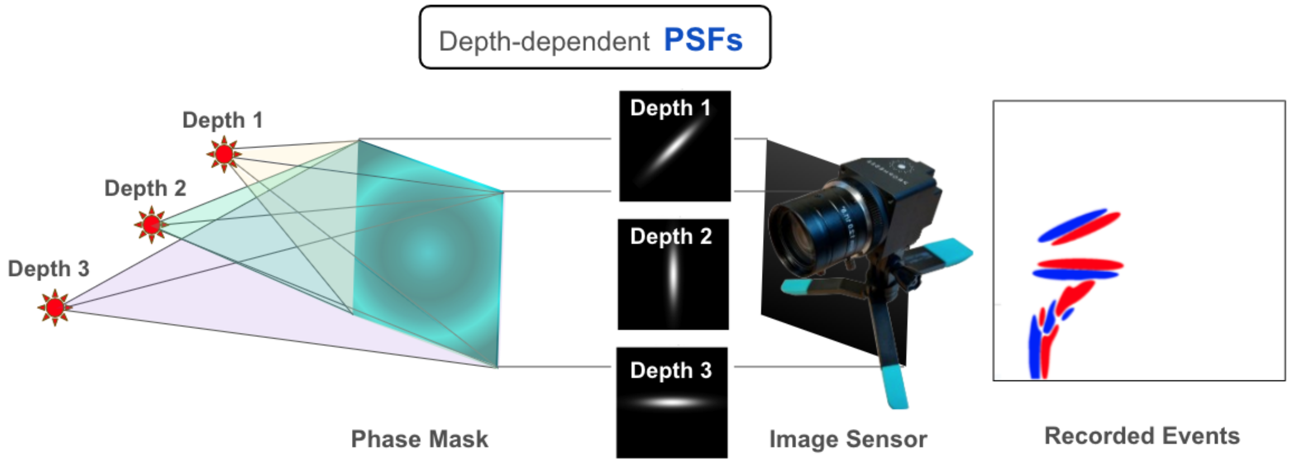

Figure 1: Overview of event-based motion detection using an optimized phase mask with depth-dependent PSFs.

Abstract

We propose a new event-based 3D motion detection technique that incorporates depth-dependent point spread functions (PSFs) into event cameras. Our technique is based on the event-camera vision, where the specialized imaging sensors are used to take advantage of the pixel-wise brightness changes. Each pixel maintains its own reference brightness level, ensuring robustness against motion blur and achieving high sparsity. By introducing a phasemap that generates depth-dependent PSFs, event cameras can capture 3D information rather than being limited to 2D. We demonstrate this capability through simulations of a 3D object tracking test and implement an end-to-end phase mask optimization process. This process simultaneously updates the phase mask height map and the neural network to determine the trajectory of the 3D object. Through exploratory end-to-end phase mask optimization, we verified that our produced phase mask can produce event-oriented, depth-dependent PSFs, potentially paving the way for full 3D motion detection using event cameras.

CS 6662 Computational Imaging Final Report, Fall, 2024, Cornell University
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Adam Cahall and Haley Lee. 2024. Event-based 3D Motion Detection with Depth-Dependent PSFs. In *Proceedings of Computational Imaging Final Report (CS 6662 Computational Imaging Final Report).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

In recent years, event cameras have emerged as an exciting new class of imaging sensors. Inspired by the human eye, each pixel in the event camera operates separately, and stores its own reference brightness level. Whenever the log intensity changes at a pixel beyond a threshold, an “event” is recorded, which contains the pixel location and the polarity of the change (i.e. whether the brightness increased or decreased). Event cameras have several key advantages over traditional CMOS sensors, including high sparsity, high temporal resolution, and low power requirements [3]. Due to these advantages, event cameras have been applied in a variety of areas, including robotics [7] and eye-tracking [1].

In many of these areas of application, detecting depth is of key importance (e.g., for object avoidance in drones [2]). While many techniques have been proposed for detecting depth with event cameras (to be discussed in Related Work), one strategy that has been relatively under-explored is point spread function (PSF) engineering. Using a phase or amplitude mask, PSF engineering allows for the PSF of an imaging system to be manipulated such that it changes with depth, thus encoding depth information into captured images (or events). In this paper, we explore the potential of PSF engineering for event-based depth detection, specifically focusing on the 3D object tracking task. Our contributions are as follows:

- We demonstrate the strengths and weaknesses of a simple hand-crafted event-based PSF engineering technique in a simulated 3D object tracking environment.
- We develop a more sophisticated end-to-end optimization technique which jointly learns depth-dependent PSFs (via a parameterized phase mask) and optimized weights for a 3D object tracking convolutional neural network. We show preliminary results that demonstrate this technique's ability to learn depth-dependent PSFs.

2 Related Work and Background

2.1 Depth Detection in Event Cameras

Due to the need for depth information in application areas, many event-based depth detection techniques have been proposed. Using a stereo pair of event cameras, works such as [8] have attempted to match the two event streams to predict disparities (and thus depth). Another work, [6], uses a conventional camera paired with a standard in a stereo setup in a "best of both worlds" approach. The obvious downside of these approaches is the requirement of two synchronized imaging sensors. Other works address this, and instead propose monocular approaches which use a single event camera. For instance, [11] uses stereo event camera data for training, but a single event camera for prediction, following an approach similar to Monodepth ([13]). Alternatively, [12] instead uses conventional camera training data in a self-supervised approach to monocular event-based depth estimation.

Overall though, the existing technique most related to ours is [9], as it is the only other paper, to our knowledge, that explores PSF engineering for depth detection in event cameras. However, their approach differs significantly than ours, as they opt to use an information-theoretic method to learn the phase mask separately from learning to predict depth, whereas we use an end-to-end joint optimization technique.

2.2 PSF Engineering

PSF engineering is the process of designing a phase mask to manipulate the Point Spread Function (PSF), which describes how a point source of light is spread out by an imaging system. By carefully engineering the phase mask, we can create a depth-dependent PSF, denoted as $h(x, y, z)$, where the PSF changes based on the 3D position (x, y, z) of the object. When a scene is captured, the light from objects at depth z is propagated through the system, producing a PSF that varies with the object's depth. In a pinhole projection model, the intensity of light at the image plane $I_t(u, v)$ at coordinates (u, v) can be expressed as:

$$I_t(u,v) = \delta\left(u - \frac{f}{z}x(t), v - \frac{f}{z}y(t)\right),$$

where $x(t)$ and $y(t)$ describe the object's trajectory, and f is the focal length. The captured image at the sensor is the convolution of the depth-dependent PSF $h_z(t)$ and the projected image I_t, expressed as:

$$\left[h_{z(t)} * I_t\right](u,v) = h(x(t), y(t); z(t)).$$

Here, the depth-dependent PSF h_z inherently encodes the 3D position of the object into the captured image. By incorporating PSF engineering into the imaging system, we can directly encode depth information (z) into the events generated by an event camera. This approach allows us to recover not only the object's position in the $x - y$ plane but also its depth z, enabling efficient and accurate 3D motion detection.

3 Methods & Experimental Details

3.1 3D object tracking simulation

We evaluated the effectiveness of PSF engineering for event cameras on several variations of a simulated 3D object tracking task. Our setup is quite simple, consisting of a single ball moving inside an otherwise empty 200 x 200 x 200 box. For our initial experiments, we ignore the phase mask component of the setup and use handcrafted depth-dependent PSFs for simplicity. Our chosen PSFs rotate from 0 to 90 degrees as depth increases from 0 to 200.

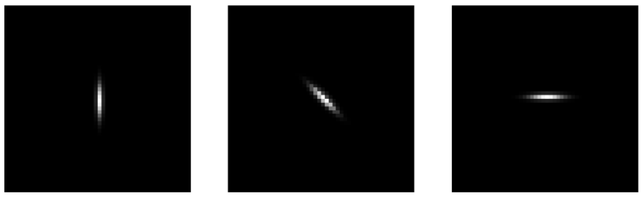
Figure 2: PSFs from depths z=0, z=100, and z=200 (enlarged for clearer visualization).

In order to generate simulated depth-encoded binned event frames, we use our forward model. Since the only object moving in 3D space in our scene is the singular ball (which we further assume lies at a single depth), we only need to convolve our input image with a single PSF; namely, the one that corresponds to the current depth of the ball. To generate events given both a video of the ball moving along a trajectory and the ground-truth depths, we first take the difference between two consecutive frames after they have been convolved with the correct PSFs. Then, we are able to use a brightness change threshold to determine all events that occurred between the two frames.

To track these 3D trajectories, we make use of a Convolutional Neural Network (CNN). Our CNN takes as input a 200 x 200 event frame, and outputs a 3D coordinate vector in $\mathbb{R}^3$. Each input event frame is first passed through 4 convolutional blocks, each containing a convolutional layer, a ReLU activation function, and a max pooling layer. The output of the final convolutional block is then flattened into a vector and processed by two fully-connected layers separated by a ReLU in order to generate a final predicted 3D location.

We make use of a two-component loss function when training our CNN model that contains both a standard mean-square-error

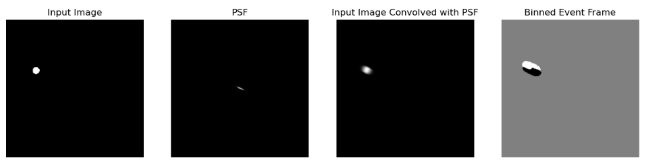

Figure 3: Visualization of the binned event frame formation process. The second input image is omitted as it appears quite similar to the first (due to small movement between frames), but we can see based on the event frame that the ball appears to be moving upwards (here white corresponds to positive events and black corresponds to negative events).

(MSE) component and a temporal consistency penalty (of our own design):

$$\text{Batch Loss} = \frac{1}{n}\sum_{i=1}^{n}||y_i - \hat{y}_i||_2^2 + \lambda \cdot \frac{1}{n-1}\sum_{i=2}^{n}||\hat{y}_i - \hat{y}_{i-1}||_2^2$$

where the first term is the MSE and the second term is the temporal consistency penalty. Here, n is the batch size, y_i is the ground-truth 3D location vector, $\hat{y}_i$ is the predicted 3D location vector, and λ is a hyperparameter that controls how heavily the temporal consistency penalty is weighted. We find that $\lambda = 0.1$ works well for our experiments.

The temporal consistency penalty that we introduce specifically penalizes large jumps in consecutive 3D location predictions (note that this loss relies on the fact that our batches contain event frames from only a single trajectory). The reasoning behind introducing this penalty was that during our initial runs, we noticed that our model would generally track the ground-truth path of the ball effectively, but on occasion would make an outlier prediction a significant distance away from the ground-truth location (and the predicted location at the previous time step). After adding the temporal consistency penalty to our overall loss function, we found that qualitatively, our predicted trajectories were smoother-looking, and quantitatively, our final test MSEs were lower. Since during inference, our CNN makes 3D location predictions for each event frame separately (and thus has no way to directly enforce small jumps between frames), it is quite surprising that this temporal consistency term is still somehow effective at teaching the model to learn to not make these outlier predictions. In the future, we are interested in investigating this phenomenon further.

We train and test our CNN model in two variations of our simulated ball-tracking environment in order to assess the potential effectiveness of PSF engineering in the event-based 3D tracking domain. We begin with an idealized setup: the "Easy Setting", in which each trajectory in both the train and test datasets contains a identically-shaped ball with identical intensity and size, and no noise.

In the (relatively) "Hard Setting", we introduce significant variations across trajectories. Specifically, before generating each trajectory in the training set, we introduce the following perturbations:

- **Varying Shape:** We set the object to be a regular n-sided polygon, where n is randomly-chosen integer in {3, 4, 6, 7} (note the exclusion of 5– this is intentional, and the reasoning is discussed below).
- **Varying Size:** We randomly set the radius of the object to some integer value in the range [1, 25].
- **Added Noise:** We initially set the intensity of the object to a uniform value (0.5), and then add Gaussian noise with mean 0 and variance 0.05 to each pixel's intensity.

For testing, we generate two separate evaluation datasets: one that contains trajectories generated using the procedure described above, and another containing trajectories of only 5-sided objects (but with the same size and noise variations as above). The purpose of the second test set is to evaluate the generalization capability of our model, as 5-sided objects were held out from our training set.

For both of the above environments, we create a training dataset of binned event frames from 150 Brownian motion trajectories that are each 129 time steps long (yielding a total of $150 * 128 = 19200$ event frames). In each trajectory, the object is assigned a random starting position, with each starting coordinate being chosen randomly from $[2r, 200\text{-}2r]$ (where r is the radius of the object) in order to reduce the likelihood of the object getting stuck along the border of the box. We also create a validation dataset using the event frames from a set of 15 trajectories generated in an identical manner.

We use the Adam optimizer with a learning rate of 0.001, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ to train our CNN model for 50 epochs. At the end of each epoch, we compute the current model's loss on the validation set. We set our final trained model to be the version from the best-performing epoch.

For evaluation in the Easy Setting, we tested our final trained model on a set of 50 128-event-frame-long trajectories, and computed a variety of summary statistics based on the MSE of each trajectory (contained in the Results section). In the Hard Setting, we test our model on two sets of trajectories: one of objects generated using the same procedure as the training set, and another containing 5-sided objects (recall that 5 is the held-out side length from the training data). Lastly, for both settings, we also generated videos of the CNN's tracking performance on several trajectories to analyze their effectiveness qualitatively. Example videos are included in the presentation recording.

3.2 End to end phase mask optimization

In the previous section, we verified the performance of the handcrafted depth-dependent PSFs recorded with events. However, in practice, to take advantage of a spatial light modulator (SLM) and build a compact imaging setup, we need to produce a phasemap that can be displayed on the SLM. This phasemap is then required to generate depth-dependent PSFs. Therefore, we attempt to build an end-to-end optimization pipeline. First, the dataset is provided in a 4D format that includes the spatial location (x, y), the depth (z), and the time factor (t). To produce events, where the intensity difference between adjacent time frames is calculated, we start with two sets of objects: at time t and $t+1$. Using the ground truth depth z, each object is propagated over a distance z. For this propagation,

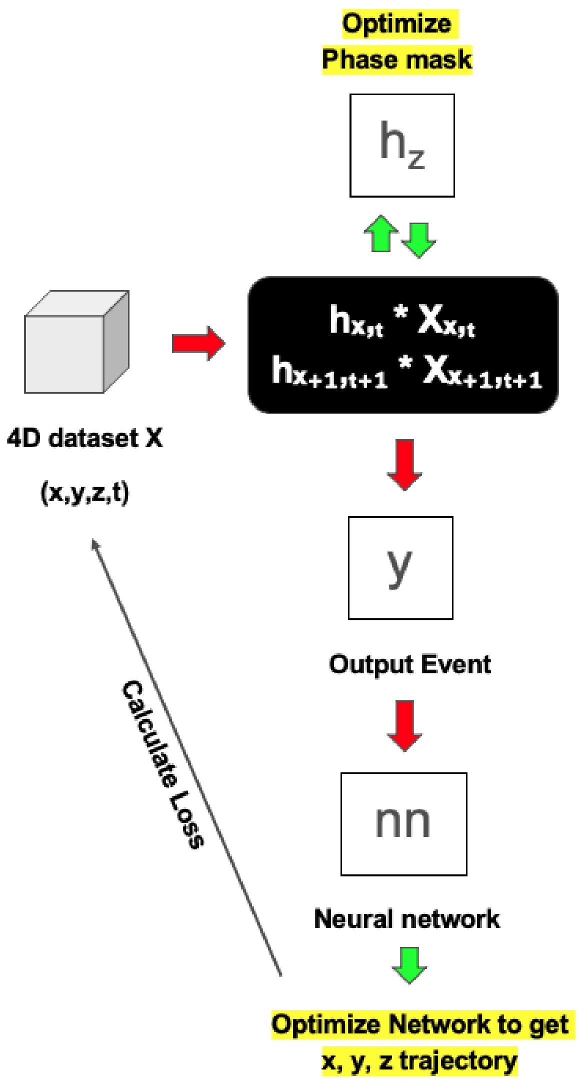

Figure 4: Pipline of end to end phase mask optimization

we use the angular spectrum method:

$$H(f_x, f_y, z) = \exp\left(ikz\sqrt{1-\lambda^2(f_x^2+f_y^2)}\right), \quad \text{where} \quad k = \frac{2\pi n}{\lambda}.$$

Here, H represents the transfer function in the frequency domain, f_x, f_y denote spatial frequencies, and k is the wave number. To properly propagate the objects, they must first be transformed into the Fourier space and multiplied with the transfer function corresponding to a particular z value. The inverse Fourier transform is then applied to return the objects to the spatial domain. Next, each result is convolved with the phasemap. Events are subsequently created based on these convolved results. This process accounts for what would occur in a real imaging system. In such a system, light reflected from objects at different depths travels varying distances z towards the SLM (spatial light modulator). When the light reaches the SLM, the optimized phase map displayed on the SLM modifies the wavefront. The resulting light is recorded as an event upon arrival at the sensor. These events are then fed into the CNN model mentioned above. The network's final task is to determine the exact trajectory and depth of the objects.

The entire pipeline is illustrated in Figure 4, where each iteration updates both the phase map and the neural network model. As mentioned in the previous section, we used the Adam optimizer; however, this time we applied different learning rates for optimizing the phase map and the network. We introduced a class called MyEnsemble, which enables joint optimization of the phase map and the network. This setup allows gradients to flow through both modules during backpropagation, ensuring that they are updated simultaneously. For the learning rates, we set $l_1 = 0.01$ and $l_2 = 0.001$ for the phasemap and for the network.

We used Zernike polynomials to initialize the phase map [10]. The phase map is constructed as a weighted sum of these polynomials, with the coefficients serving as adjustable parameters:

$$\text{Phase Map}(x, y, r) = \sum_{i=0}^{28} c_i Z_i(x, y, r),$$

Here, $Z_i(x, y, r)$ represents the Zernike polynomials that describe specific phase aberrations, c_i are the corresponding coefficients that control the amplitude of each polynomial, x, y denote the spatial coordinates, and $r = \sqrt{x^2+y^2}$ is the radial coordinate. This initialization provides the starting point for the optimization process, ensuring that the phase map is interpretable and efficient to refine during training.

4 Results

4.1 3D object tracking simulation

Mean	Median	Minimum	Maximum	Standard Deviation
3.0	2.7	2.3	15	0.8

Table 1: MSE Statistics over 50 Test Trajectories (Easy Setting)

	Mean	Median	Min	Max	Standard Deviation
Familiar objects	7.4	5.3	2.7	24.6	5.6
Unfamiliar objects	10.4	7.5	4.7	98.5	13.2

Table 2: MSE Statistics over 50 Test Trajectories (Hard Setting)

x	y	z
0.9	1.0	3.3

Table 3: Average Absolute Error in Each Coordinate (Easy Setting)

Overall, our CNN model trained in the Hard Setting struggles significantly more than our model trained in the Easy Setting. When being evaluated on trajectories of familiar objects (i.e., ones with

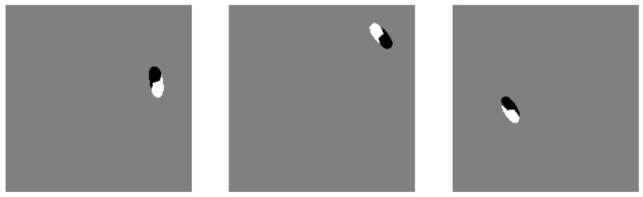

Figure 5: Example events from the Easy Setting.

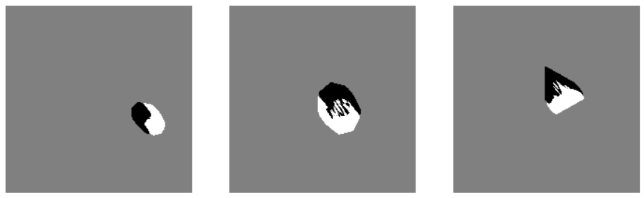

Figure 6: Example events from the Easy Setting.

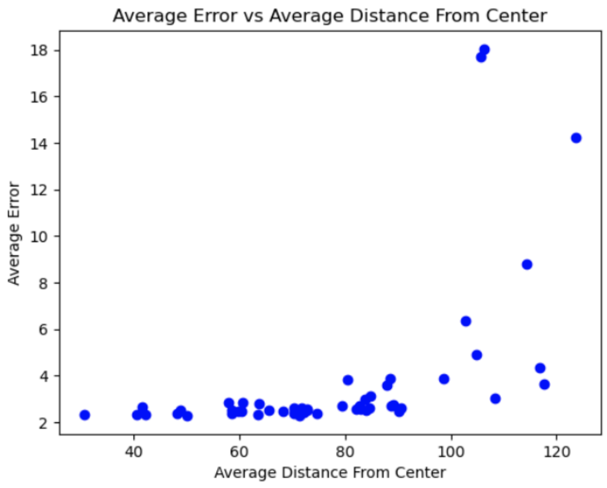

Figure 7: Average Error vs Average Distance From Center, evaluated over 50 trajectories in the Easy Setting.

the same side lengths as those in the training set), the model still obtains over twice the mean MSE and five times the standard deviation as the model being trained and evaluated in the Easy Setting due to still being subject to varying size and noise in the objects. When evaluated on unfamiliar objects (i.e., 5-sided), the average MSE unsurprisingly further increases, to over three times as in the Easy Setting. We furthermore see that there are certain trajectories in this unfamiliar, more difficult environment where the model fails to even roughly track the ground truth trajectory, as evident by the maximum MSE of 98.5.

By comparing the event frames captured in the Easy Setting and the Hard Setting, we can visualize their differences. The added noise and varied shapes and sizes cause the event frames in the Hard Setting to be significantly more diverse than in the Easy Setting. Additionally, it is substantially more difficult to visually determine a confident estimate of the PSF angle in the Hard Setting. As most

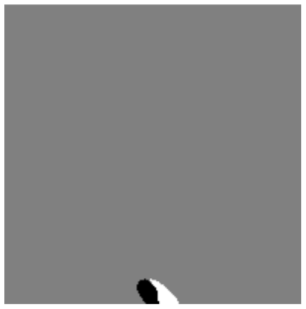

Figure 8: Example of events being cut off when object is close to the border.

of the error comes in the z direction (though Table 3 above is for the Easy Setting, the same result holds in the Hard Setting), it seems reasonable to conclude that the overall increase in error in the Hard Setting is mainly due to increased difficulties in estimating the PSF angle causing increased difficulties in estimating depth. Overall, to be able to succeed at the 3D object tracking task in a more difficult environment like the Hard Setting (and certainly for a truly realistic setting), our results show that our model needs to be improved in one or more facets. These improvements could include increasing the training set size, increasing the model complexity (or switching to a different model architecture altogether), or integrating a learned phase mask which provides potentially more optimal depth-dependent PSFs than our simple handcrafted PSFs.

An additional final trend that we observed in both environments was poorer 3D tracking performance on trajectories where the object remains near the border of the box. In our experiments, we restrict any part of the object from going out of frame, but some events still end up being cut off, as seen in Figure 6. Overall, it seems likely that it would be hard to completely avoid the diminished performance on tracking objects near the edge of the scene, though perhaps more optimal PSFs that come from a learned phase mask would be adapted in a way that somewhat mitigates this effect.

4.2 End to end phase mask optimization

After several iterations, we successfully optimized the phase map. Figure 9 shows the resulting phase map: (a) represents the phase map at the initial stage, while (b) shows the final optimized phase map after convergence. (c) and (d) display the results of the PSFs (Point Spread Functions), which were propagated from the phase map at different depths. The resulting PSFs exhibit distinct features, indicating that we have successfully created a phase map capable of generating depth-dependent PSFs. Figure 10 shows the loss graph during the training and validation process. The graph converges smoothly over time, demonstrating the appropriateness of the learning rate and the robustness of the forward model.

However, there is still room for improvement. First, the optimized phase map requires further inspection. During the iteration process, the phase map did not change significantly. We still need to analyze how each Zernike coefficient evolves with each iteration. Furthermore, the logic behind how the initial Zernike phase map is generated also requires further investigation.

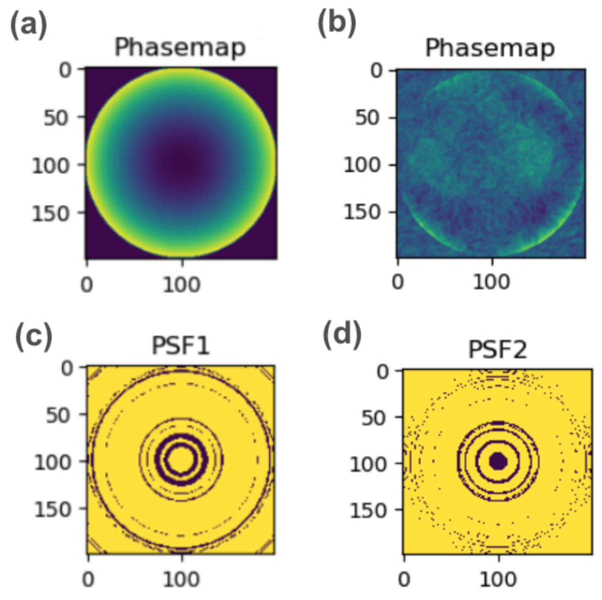

Figure 9: Phase mask optimization results: (a) Intial phase map (b) Optimized phase map (c) Propagated PSF with depth z_1 (d) Propagated PSF with depth z_2

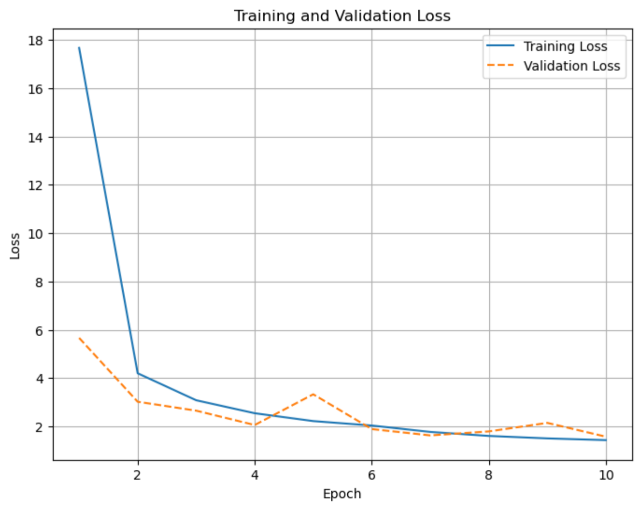

Figure 10: Loss graph of phase map optimization

Regarding the PSFs, although they appear completely distinct, their values are extremely small, on the order of 10^{-8}. To ensure that the forward model is correct, we need to carefully examine the phase map even before optimization. Our initial plan is to test a phase map in the form of a lens and verify whether it can focus to a point.

5 Conclusion and Future Work

We reported a method to detect event-based 3D motion where depth-dependent PSFs are employed. The integration of PSF engineering and event vision allows to encode the 3D real-world scene into event data. Our technique enables us to reconstruct the depth from a single camera, enabling efficient and novel imaging atmosphere.

Overall, our results demonstrate the potential of PSF engineering in event-based 3D motion detection. Though our initial hand-crafted PSF approach struggled when evaluated in a more complex environment, we believe that by instead optimizing our CNN jointly with the phase mask in the proposed end-to-end scheme, we could learn more effective PSFs and succeed in the Hard Setting.

Once we are able to achieve satisfactory performance in the Hard Setting, we plan to move on to using a real-world dataset for our experiments. If we decide to continue focusing on 3D object tracking, there exist many open source datasets for this task, such as KITTI [5], which we could make use of. Since KITTI and most other existing datasets contain RGB camera images rather than event streams, we plan to use an existing video-to-event simulator ([4]) to simulate events. Additionally, we will use our forward model to simulate our phase mask. The end goal will be to physically realize our learned phase mask using an SLM, and evaluate our 3D object tracking system in the real world.

6 Contributions

A.C and H.L contributed equally to this work. A.C and H.L both devised and discussed the concept of event-based 3D motion detection technique. A.C mainly built the convolutional neural network and simulated the 3D object tracking simulation. H.L mainly developed and optimized phasemap. All the authors discussed the experiment results and contributed to the writing of the paper.

References

[1] Anastasios N. Angelopoulos, Julien N.P. Martel, Amit P. Kohli, Jo rg Conradt, and Gordon Wetzstein. 2022. Event-Based Near-Eye Gaze Tracking Beyond 10,000 Hz. *IEEE Transactions on Visualization and Computer Graphics.* (2022).
[2] DAVIDE FALANGA and KEVIN KLEBER andDAVIDE SCARAMUZZA. 2020. Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics* 5 (2020).
[3] Guillermo Gallego, Tobi Delbru ck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J Davison, Jo rg Conradt, Kostas Daniilidis, and et. al. 2020. Event-based vision: A survey. *IEEE transactions on pattern analysis and machine intelligence* 44, 1 (2020), 154–180.
[4] Daniel Gehrig, Mathias Gehrig, Javier Hidalgo-Carrio , and Davide Scaramuzza. 2020. Video to Events: Recycling Video Datasets for Event Cameras. *Conference on Computer Vision and Pattern Recognition* (2020).
[5] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. 2013. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research* (2013).
[6] S. Mohammad Mostafavi I., Kuk-Jin Yoon, and Jonghyun Choi. 2021. Stereo Depth from Events Cameras: Concentrate and Focus on the Future. *International Conference on Computer Vision* (2021).
[7] Craig Iaboni, Himanshu Patel, Deepan Lobo, Ji-Won Choi, and Pramod Abichandani. 2021. Event camera based real-time detection and tracking of indoor ground robots. *IEEE Access* (2021).
[8] Yeongwoo Nam, Mohammad Mostafavi, Kuk-Jin Yoon, and Jonghyun Choi. 2022. Stereo Depth from Events Cameras: Concentrate and Focus on the Future. *Conference on Computer Vision and Pattern Recognition* (2022).
[9] Sachin Shah, Matthew A. Chan, Haoming Cai, Jingxi Chen, Sakshum Kulshrestha, Chahat Deep Singh, Yiannis Aloimonos, and Christopher A. Metzler. 2024. CodedEvents: Optimal Point-Spread-Function Engineering for 3D-Tracking with Event Cameras. *Conference on Computer Vision and Pattern Recognition* (2024).
[10] Yicheng Wu, Vivek Boominathan, Huaijin Chen, Aswin Sankaranarayanan, and Ashok Veeraraghavan. 2019. PhaseCam3D — Learning Phase Masks for Passive Single View Depth Estimation. *IEEE International Conference on Computational Photography* (2019).
[11] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. 2018. Unsupervised Event-based Learning of Optical Flow, Depth, and Egomotion. *Conference on Computer Vision and Pattern Recognition* (2018).

[12] Junyu Zhu, Lina Liu, Bofeng Jiang, Feng Wen, Hongbo Zhang, Wanlong Li, , and Yong Liu. 2023. Self-supervised Event-based Monocular Depth Estimation using Cross-modal Consistency. *International Conference on Intelligent Robots and Systems* (2023).

[13] Cle ment Godard Oisin Mac Aodha Gabriel J. Brostow. 2017. Unsupervised Event-based Learning of Optical Flow, Depth, and Egomotion. *Conference on Computer Vision and Pattern Recognition* (2017).