

Event-based Vision

Contents

1	Introduction	1
2	Event Camera Fundamentals	1
2.1	Event Generation	1
3	Exercise Framework	2
3.1	Data Structure	2
3.2	Event Visualization	2
3.2.1	2D Event Visualization	2
3.2.2	3D Spatial-Temporal Visualization	2
4	Contrast Maximization Theory	2
4.1	Motion Model	2
4.2	Objective Functions	3
5	Implementation Tasks	3
5.1	Task 0: Data Loading	3
5.2	Task 1: Event Visualization	3
5.3	Task 2: Contrast Maximization	4

1 Introduction

Event cameras represent a paradigm shift in computer vision sensing. Unlike conventional cameras that capture intensity frames at fixed time intervals, event cameras detect pixel-level brightness changes asynchronously. This exercise explores the processing and analysis of event-based data through contrast maximization techniques, focusing on motion estimation and event visualization.

2 Event Camera Fundamentals

2.1 Event Generation

An event camera generates an event e_k when the logarithmic brightness change at a pixel (x, y) exceeds a threshold C :

$$|\log(I(t)) - \log(I(t - \Delta t))| \geq C \quad (1)$$

Each event e_k is represented as a tuple:

$$e_k = (x_k, y_k, t_k, p_k) \quad (2)$$

where:

- (x_k, y_k) is the pixel location

- t_k is the timestamp (in microseconds)
- $p_k \in \{-1, +1\}$ is the polarity indicating brightness decrease/increase

3 Exercise Framework

3.1 Data Structure

Events are stored in four synchronized NumPy arrays:

$$\begin{aligned}
 \mathbf{x} &\in \mathbb{R}^N : \text{x-coordinates} \\
 \mathbf{y} &\in \mathbb{R}^N : \text{y-coordinates} \\
 \mathbf{t} &\in \mathbb{R}^N : \text{timestamps} \\
 \mathbf{p} &\in \{-1, +1\}^N : \text{polarities}
 \end{aligned} \tag{3}$$

3.2 Event Visualization

The framework implements three key visualization methods:

3.2.1 2D Event Visualization

Events are projected onto the image plane with:

- Blue points for positive events ($p_k = +1$)
- Red points for negative events ($p_k = -1$)
- Proper image coordinates (y-axis inverted)
- Adjustable scatter point size for density visualization

3.2.2 3D Spatial-Temporal Visualization

Events are plotted in a 3D space where:

- X-Y plane represents spatial coordinates
- Z-axis represents time
- View orientation set to make image plane parallel to the screen
- Aspect ratio maintained for the proper spatial-temporal relationship

4 Contrast Maximization Theory

4.1 Motion Model

The framework employs a linear velocity warp function to model motion in the event stream. This function is mathematically expressed as:

$$W(x, y, t; \theta) = \begin{bmatrix} x + v_x(t - t_{ref}) \\ y + v_y(t - t_{ref}) \end{bmatrix} \tag{4}$$

The motion model is parameterized by the velocity components (v_x, v_y) , which together form the parameter vector $\theta = [v_x, v_y]$ that will be optimized. The reference timestamp t_{ref} serves as the temporal anchor point for the motion compensation. This linear model, while simple, effectively captures the first-order motion present in many real-world scenarios.

4.2 Objective Functions

The framework implements several objective functions, each designed to capture different aspects of event alignment quality. The variance objective function serves as a fundamental measure, maximizing the spatial variance of warped events. It is mathematically formulated as:

$$J_{var}(\theta) = \frac{1}{N} \sum_{x,y} (I(x, y; \theta) - \mu)^2 \quad (5)$$

where the warped event image $I(x, y; \theta)$ is computed through the accumulation of warped events:

$$I(x, y; \theta) = \sum_k \delta(x - x'_k) \delta(y - y'_k) \quad (6)$$

with (x'_k, y'_k) representing the warped coordinates of each event.

Building upon this foundation, the R1 objective function focuses on maximizing local event density. This approach is particularly effective in scenarios with clear motion patterns:

$$J_{R1}(\theta) = \sum_{x,y} I(x, y; \theta)^2 \quad (7)$$

The Sum of Events (SoE) objective provides yet another perspective on event alignment, measuring the overall coherence of the warped event distribution:

$$J_{SoE}(\theta) = \sum_{x,y} |I(x, y; \theta)| \quad (8)$$

5 Implementation Tasks

5.1 Task 0: Data Loading

The first step in the exercise is to load the event data from the provided `circle_events.h5` file. Then you could run the full pipeline using the `python events_cmax.py -YOUR_PATH_T0/circle_events.h5` command.

5.2 Task 1: Event Visualization

The visualization process begins by creating a 3D figure with dimensions that maintain the proper aspect ratio between spatial and temporal coordinates. Events are then separated into two groups based on their polarity, with positive and negative events handled independently. Each event is plotted in the 3D space using distinct color coding - typically blue for positive events and red for negative events - to provide clear visual differentiation. The view angle is carefully adjusted to ensure the image plane appears parallel to the screen, making the spatial relationships easily interpretable. Finally, the axes are configured with appropriate labels and scales to provide context for the spatial coordinates and temporal dimension. To summarize, implement the visualization functions: `visualize_events(xs, ys, ts, ps, img_size)`

- Create a 3D figure with a proper aspect ratio
- Separate positive and negative events
- Plot events with color coding
- Set view angle for parallel image plane
- Configure axes and labels

5.3 Task 2: Contrast Maximization

After playing around with the visualization of the event camera data, you are then tasked to implement the optimization pipeline:

```
optimize_contrast(xs, ys, ts, ps, warp, obj)
```

- Initialize parameters θ_0
- Define objective function $J(\theta)$
- Compute gradients (numeric or analytic)
- Optimize using BFGS or other methods
- Return optimal parameters θ^*