

# Egocentric Event-Based Vision for Ping Pong Ball Trajectory Prediction

## Supplementary Material

### 8. The aerodynamics model of a ping-pong ball

In this section, we provide an overview of the aerodynamics model of a ping pong ball used in the paper, focusing on the equations of motion that describe its trajectory. A standard ping-pong ball moving through the air experiences four primary forces: gravitational force ( $F_g$ ), buoyancy force ( $F_b$ ), drag force ( $F_d$ ), and Magnus force ( $F_m$ ). For our investigation, we can ignore the buoyancy force  $F_b = -m_b g$ , because the mass of the air displaced by the ping-pong ball is negligible with respect to the mass of the ball  $m$ . We also neglect the spin of the ball (Magnus force component  $F_m$ ), cause it is not directly observed by the vision system due to the small dimension of the ball. Therefore, the sum of the forces acting on the ball can be expressed as  $\sum \mathbf{F} = \mathbf{F}_g + \mathbf{F}_d$ . The gravitational force is given by  $\mathbf{F}_g = -mg$ , where  $m$  represents the mass of the ball, and  $g = [0, 0, -9.81]^T$  is the acceleration due to gravity. The drag force, which opposes motion through the air, follows the equation  $\mathbf{F}_d = -\frac{1}{2}C_d\rho A|\mathbf{v}(t)|\mathbf{v}(t)$ , where  $C_d$  is the drag coefficient,  $\rho$  is the air density,  $A$  is the cross-sectional area of the ball and  $|\mathbf{v}(t)|$  is the magnitude of the velocity vector  $\mathbf{v}(t)$ . By substituting these forces, we obtain:

$$\sum \mathbf{F} = mg - \frac{1}{2}C_d\rho A|\mathbf{v}(t)|\mathbf{v}(t) \quad (11)$$

This simplifies the equation of motion to:

$$\dot{\mathbf{v}}_k(t) = g - k_d|\mathbf{v}(t)|\mathbf{v}(t) \quad (12)$$

where we set  $k_d = \frac{C_d\rho A}{2m}$ . For a standard ping-pong ball, the known values are:  $\rho = 1.225\text{kg}/\text{m}^3$ ,  $r = 0.02\text{m}$ ,  $m = 0.0027\text{kg}$  and  $C_d = 0.4$ . We additionally model the motion of the ball by introducing a simplified bouncing model. When the estimated  $z$ -coordinate of the ball is lower than  $h_{\text{table}}$  (determined using ArUco marker detection, as shown in Fig. 1) and  $\mathbf{v}_z$  is negative, we switch to the bounce model:

$$\mathbf{v}_z^+ = e \mathbf{v}_z^-, \quad \text{with } 0 < e < 1.$$

Here,  $\mathbf{v}_z^-$  represents the velocity component along the vertical axis just before impact, and  $\mathbf{v}_z^+$  is the velocity just after. This model accounts for energy loss upon impact due to inelastic collisions with the table.

#### 8.1. Including Rotational Dynamics

A more precise representation of the ball's motion should account for its rotational dynamics, particularly the Magnus force,  $F_m$ , which influences the ball's trajectory. This force

arises due to the interaction between the ball's spin and the surrounding air, significantly affecting its movement, and it is defined as follows:

$$F_m = C_m\rho A r(\omega \times v) \quad (13)$$

where  $C_m$  is the Magnus coefficient and  $\omega$  is the angular velocity of the ping-pong ball. The equation of motion including the Magnus force would then become:

$$\dot{\mathbf{v}}_k(t) = g - k_d|\mathbf{v}(t)|\mathbf{v}(t) + k_m(\omega \times v) \quad (14)$$

or in its discrete formulation:

$$\mathbf{v}(t_i) = \mathbf{v}(t_{i-1}) + \begin{bmatrix} -k_d\|\mathbf{v}\| & -k_m\omega_z & k_m\omega_y \\ k_m\omega_z & -k_d\|\mathbf{v}\| & -k_m\omega_x \\ -k_m\omega_y & k_m\omega_x & -k_d\|\mathbf{v}\| \end{bmatrix} \mathbf{v}(t_{i-1})\Delta t + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \Delta t \quad (15)$$

where  $k_m = \frac{C_m\rho A r}{m}$ . When incorporating rotational dynamics into the motion model, setting the initial conditions of the differential equation requires also specifying an initial estimate of the ball's angular velocity. However, this quantity is not directly measurable from our observations, but it can be inferred from the trajectory data  $\{t_{\mathcal{B}_k}, \hat{\mathbf{p}}_k, \hat{\mathbf{v}}_k\}_{k=1\dots K}$  estimated from the measurements.

### 9. Sensing Latency Analysis

We present an analysis of the sensing latency of our algorithm, which refers to the time window required to detect motion events and produce reliable results. As described in [6], an obstacle is detected using an event camera when its edges generate an event. This occurs when the relative motion between the camera and the obstacle causes a significant intensity change, triggering an event. Prior work [6] has shown that an obstacle's edge produces an event when its projection on the image plane moves by at least one pixel. As already shown in [6], the time required for an obstacle to traverse a pixel distance  $\Delta u = 1$  in the image plane is given by:

$$\tau_E = \frac{1}{\hat{\mathbf{v}}} \frac{\Delta u d^2}{f r_o + \Delta u d} \quad (16)$$

where  $\hat{\mathbf{v}}$  is the object's relative velocity with respect to the camera,  $d$  represents the obstacle's distance along the camera's optical axis,  $r_o$  is the obstacle's radius, and  $f$  is

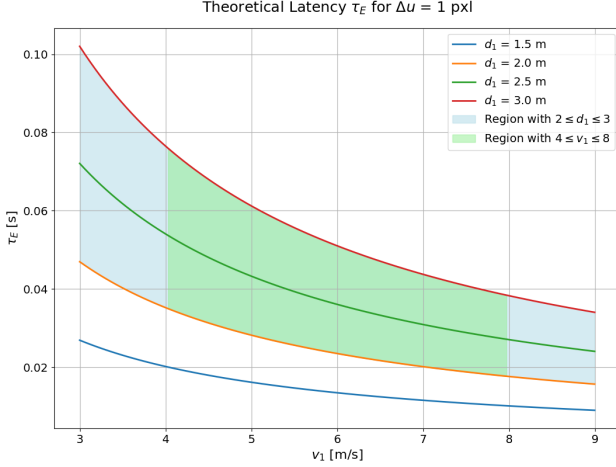


Figure 5. The sensing latency  $\tau_E$  of an event camera with  $640 \times 480$  resolution and a focal length of 6 mm. The shaded green region represents the ideal sensing latency conditions based on our dataset, where the relative velocity between the ball and the Project Aria glasses varies from approximately  $\sim 4$  m/s to  $\sim 8$  m/s.

the camera’s focal length. This calculation assumes that the optical axis passes through the geometric center of the obstacle, which is approximated as a segment. Figure 5 illustrates the theoretical sensing latency for an event camera to perceive a  $\Delta u = 1$  pixel motion in the image plane of a ping-pong ball, as a function of distance  $d$  and speed  $\hat{v}$ . In our specific case, we aim to track the ball when struck by the opponent’s racket, therefore the ball is typically observed at distances  $d$  ranging from 2 to 3 meters.

## 10. Deep Conditional Generative Network

An overview of the variational autoencoder network introduced in [16], which is employed for trajectory prediction, and the outcomes of which have been discussed in Section 5, is provided in this paragraph. The work proposes a Deep Conditional Generative Network (DCGN) for real-time trajectory prediction, mapping partial trajectories to a latent Gaussian space to predict future points. In this framework, a trajectory is represented as a probability distribution conditioned on observed data. Let  $\mathbf{x}_{1:t}$  denote the observed trajectory up to time  $t$ , and  $\mathbf{x}_{t+1:T}$  represent the future trajectory to be predicted. The model efficiently learns the conditional distribution  $p(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t}) = \int p(\mathbf{x}_{t+1:T}|\mathbf{z}, \mathbf{x}_{1:t}) p(\mathbf{z}|\mathbf{x}_{1:t}) d\mathbf{z}$ , which is achieved by introducing a latent variable  $\mathbf{z}$  that captures the underlying dynamics of the trajectory. The training procedure involves maximizing the evidence lower bound (ELBO) to approximate the true posterior distribution. This method enables better long-term prediction of complex trajectories compared to LSTMs and differential equations, thanks to its

probabilistic modeling, uncertainty estimation, and efficient latent space representation.

To assess the predictive capabilities of the DCGN model, we conducted an additional evaluation on the ground truth trajectories. By using different prediction horizon lengths  $T$ , we analyzed how well the model can forecast future motion while minimizing error. The obtained results, presented in Table 7, show that larger prediction horizons generally lead to better performance, as indicated by lower Root Mean Squared Error (RMSE) values.

Horizon Time	RMSE
0.01	0.2616
0.03	0.2055
0.12	0.1737
0.20	0.1470
0.30	<b>0.1177</b>

Table 7. RMSE values of the predicted trajectory across the entire dataset for different horizon prediction times  $T$ .

The DCGN model was trained on ground truth trajectories upsampled to 0.8 kHz, using an 80/20 split for training and validation. We observed that for short horizons, the model struggles to produce accurate predictions, resulting in high RMSE values. Figure 6 visually compares the predicted and ground truth trajectories for three selected horizon values:  $T = 0.03$  s,  $T = 0.1$  s, and  $T = 2$  s. The results demonstrate that for  $T = 0.03$  s, the predicted trajectory deviates significantly from the ground truth, aligning with the poor performance reflected in Table 7. This is also consistent with the findings in Section 5, where even worse performance was observed when evaluating on noisy measurements from the perception pipeline.

## 11. Complementary Evaluation Plots

In this section, we provide additional plots and evaluations of the entire pipeline. First, we present error plots obtained from the online trajectory prediction method, presented in Sec. 5. Figure 8 highlights the gradual improvement of the trajectory prediction as the ball is continuously tracked and its path recalculated over time, using different sample trajectories. Each curve represents a different game sequence, with variations in duration due to differences in ball detectability across sequences. The results demonstrate that increasing the accumulation time window and recomputing the trajectory with more recent measurements results in a more accurate trajectory estimate.

Figure 7, on the other hand, shows the error distribution of the predicted bouncing points on the table compared to the ground truth counterparts. The visualization is consistent with the results in Table 6, showing that the DCGM

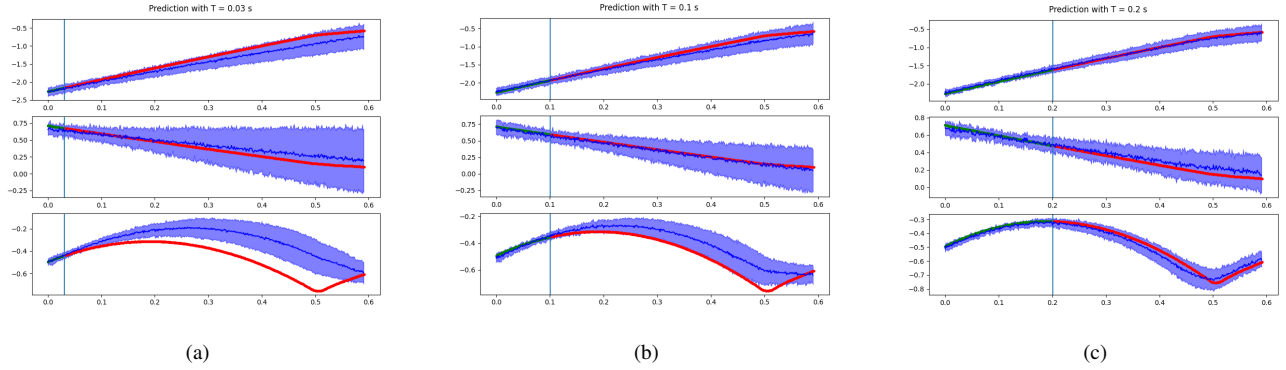


Figure 6. Visualization of the predicted trajectory compared to the ground truth for different prediction time horizon values (a)  $T = 0.03$  s, (b)  $T = 0.1$  s, and (c)  $T = 0.2$  s. The x, y, and z components of the trajectory are shown, where the green segments represent the input to the network (before the split), the red segments represent the ground truth after the split, and the blue lines indicate the predicted trajectory with the  $3\sigma$  standard deviation.

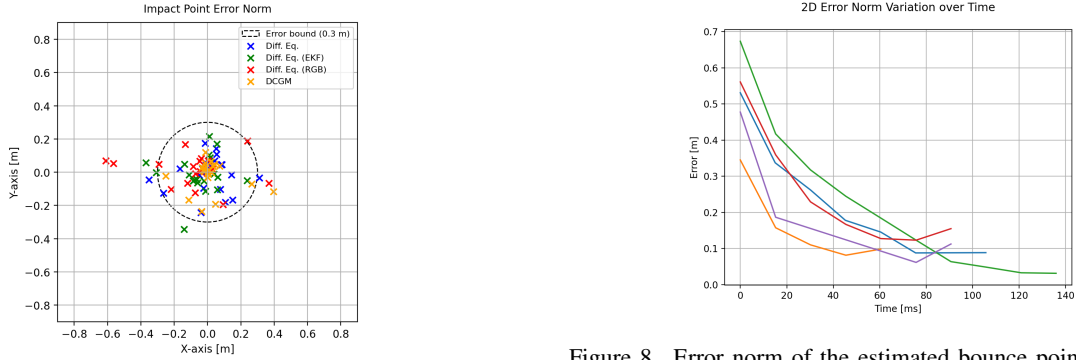


Figure 7. The plot shows the relative error of the impact point for each predicted trajectory with respect to their ground truth counterparts (each  $\times$  represents an evaluated trajectory). A boundary circle with  $r = 0.3$  m is shown as a reference.

model and the standard high-frequency updates using differential equation fitting exhibit a more concentrated distribution around the origin. In contrast, the low-framerate model fitting produces a wider spread of points.

## 12. Audio Signals Peak Detection

The evaluation of our pipeline has been carried out on sequences beginning precisely at the moment the ball impacts the opponent's racket. To segment long game sequences, we leveraged the microphone audio signals provided by the Project Aria glasses recordings, as shown in Figure 9. By monitoring these signals, a pattern of peak intensities was observed, corresponding to four events: the Project Aria user hitting the ball, the ball bouncing on the user's half of the table, the ball bouncing on the opponent's half, and the opponent hitting the ball. Each of these peaks exhibits different intensities due to their varying distances from the mi-

crophone. Specifically, the peak corresponding to the opponent's hit has the lowest intensity. To refine our analysis, we manually filtered the audio signal using signal processing techniques. First of all, to enhance the relevant audio features, we applied a high-pass Butterworth filter to remove low-frequency noise. After filtering, peaks in the audio signal were detected using the `find_peaks` function of the `scipy.signal` library, after thresholding the peaks with intensity higher than  $\frac{1}{4}$  of the magnitude of the time signal  $y(t)$ . The peak detection algorithm identifies local maxima that satisfy these conditions, ensuring that only the peaks of the opponent hitting the ball are captured. This process was repeated across multiple microphones for robust detection. For further improvements, a neural network could be trained to classify audio signals automatically. This would enable real-time detection of the opponent's racket ball hit, optimizing computational efficiency by selectively triggering the detection pipeline.

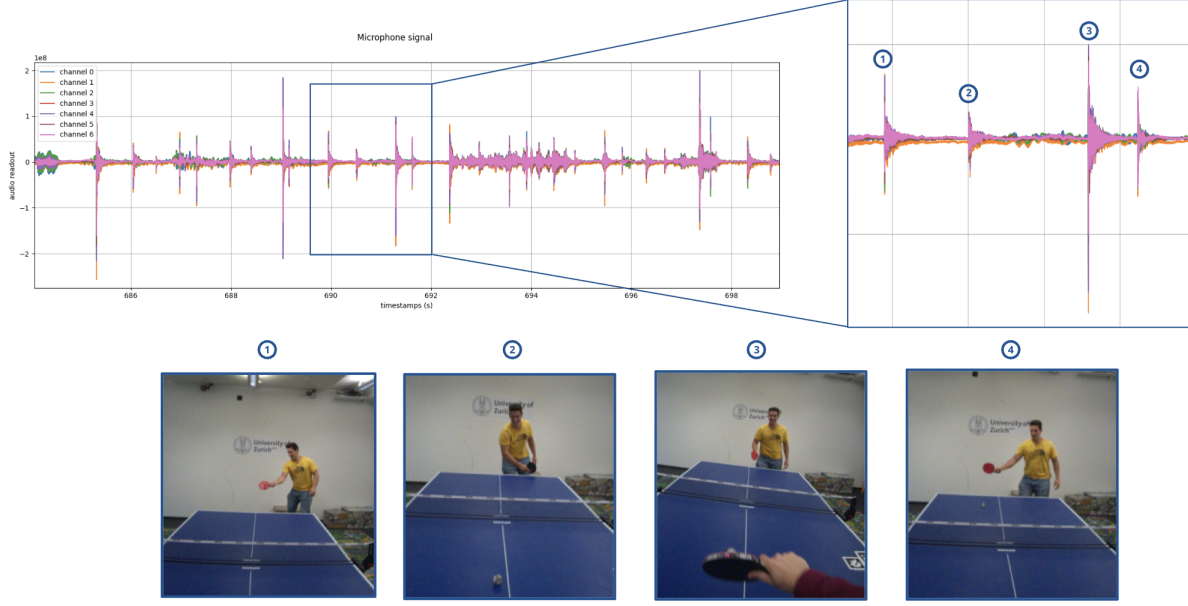


Figure 9. Microphone audio signals from a sample Project Aria glasses recording during ping-pong game.

### 13. Comparison of Circle Fitting methods

Circle fitting is a crucial component of our algorithm, as it plays a fundamental role in estimating the depth of the ball in our monocular setup. When detecting a ping pong ball at distances of up to 3 meters using a  $640 \times 480$  resolution camera, even a one-pixel error in the estimated radius can result in a depth miscalculation of several centimeters. Since depth estimation directly influences the accuracy of the x and y coordinates, such errors can significantly impact the overall pipeline. Our proposed method, described in Section 3, is compared here with two alternative approaches: ellipse fitting and circle fitting using Taubin’s method. The ellipse fitting technique determines the mean center of the detected shape and applies Principal Component Analysis (PCA) to estimate the orientation and axis lengths. The semi-major and semi-minor axes are derived from the square root of the eigenvalues of the covariance matrix, while the orientation is dictated by the principal components. On the other hand, Taubin’s method is a geometric circle fitting approach that minimizes algebraic distance while maintaining invariance to scale transformations. Figure 10 illustrates the visual results of these methods. Both alternative techniques tend to underestimate the ball’s radius, leading to inaccuracies in depth estimation. In contrast, our method remains the only reliable approach, ensuring consistent and precise measurements.

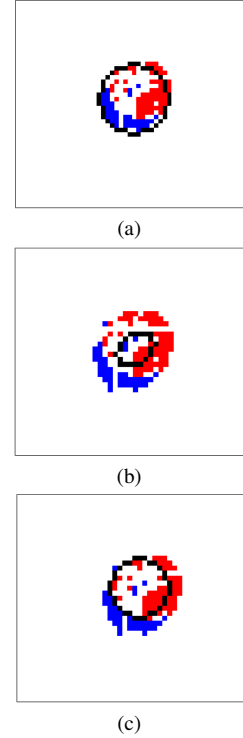


Figure 10. Side-by-side comparison of different circle fitting methods for ball detection. (a) Our proposed method, (b) ellipse fitting, and (c) Taubin’s method. The blue and red points represent positive and negative events, respectively, while the black lines indicate the estimated radius.

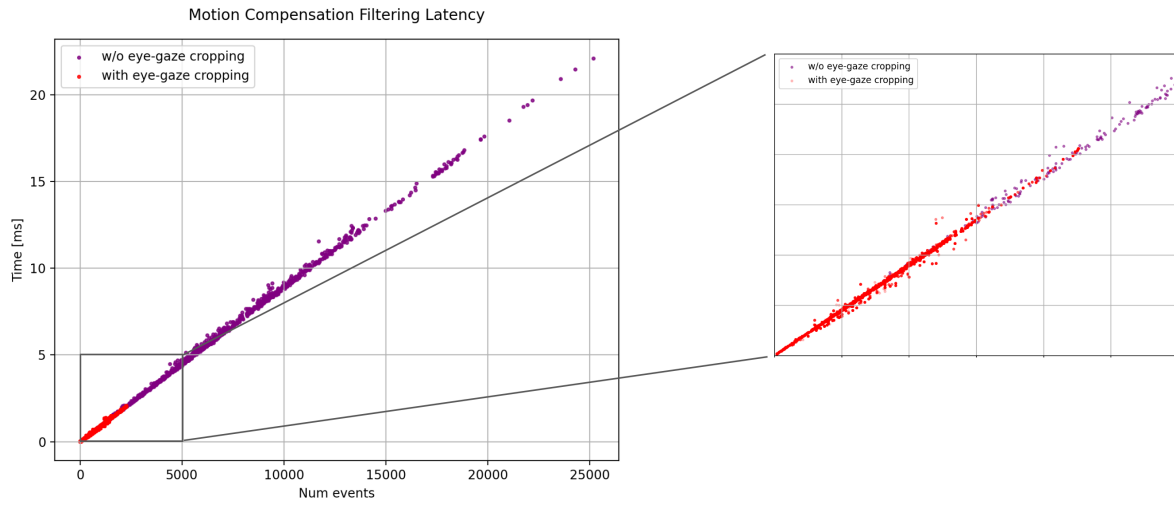


Figure 11. Time required for ego-motion compensation as a function of the number of generated events. Each dot representing a 5 ms time window of events.

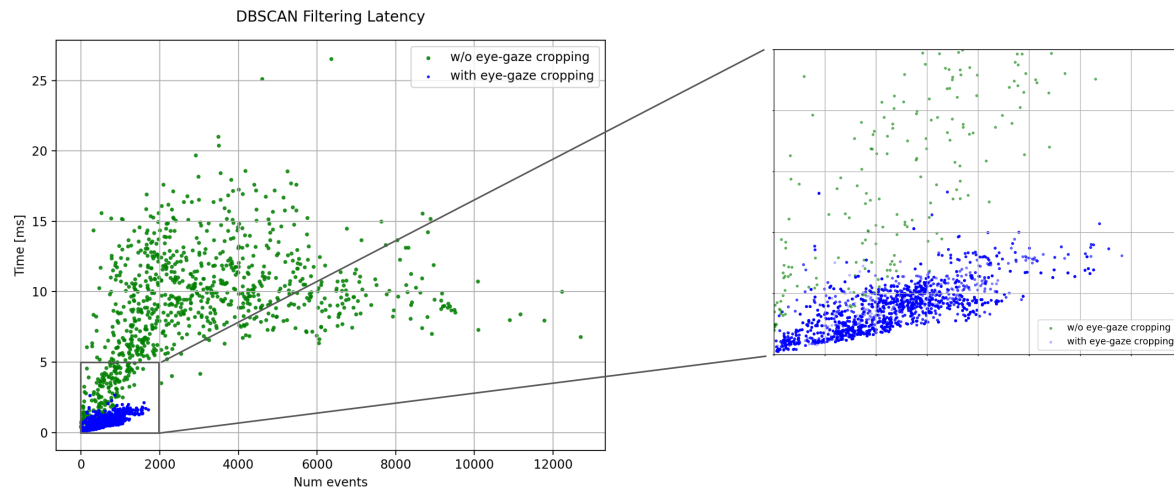


Figure 12. Time required for DBSCAN clustering of the scene's dynamic part and circularity check, based on the number of pixels from moving objects. Each dot representing a 5 ms time window of events.